Tampere University

Matti Käyrä

IMPLEMENTING AND EVALUATING NEW DIGITAL DESIGN EXCERCISES

Faculty of Information Technology and Communication Sciences
Master of Science Thesis
May 2019

ABSTRACT

Matti Käyrä: Implementing and Evaluating New Digital Design Excercises
Master of Science Thesis
Tampere University
Information Technology, Embedded Systems
May 2019

This Master's thesis is concerns digital design course that teaches the basics of digital logic. Students on course learn to design, analyze and implement combinatorial gate networks, sequential systems and state machines on both theoretical level and on practical implementations with FPGA development board.

The FPGA board that was previously used is aged and the computer exercises themselves have been fragmented between the greater course project and supplementary tasks. In this thesis, a totally new computer exercise project is designed.

Before the development of the exercises begun, the new FPGA development board had already chosen. Selection was done based on the flexibility of the system. The new board can be used by wide selection of courses to create various types of tasks. Along with the new exercise project, a MOOC-based web portal, Plussa, was taken into use. This system allows a semi-automatic grading of exercise tasks.

The objective of computer exercises is to combine the theory of digital design concepts and the practice done on paper exercises to work flow that includes real industry grade development tools. The subject of the exercise project is a game implemented on a LED matrix extension of the development board.

A reference project was designed and based on that exercise tasks were created. Exercises were divided into five separate exercise sets: Introduction to tools and development flow, arithmetic and hierarchical design, state machine design, extending existing designs and integration. On return, tasks will be graded by an automated system that fetches the students implementation, uses logic simulator on the server to verify the correct functionality and grades the return accordingly.

Without an automated grading system the grading would need to be screened manually from students screenshots. Hence, with automated system, the time spent by staff could be used more efficiently to benefit the course attendees. For students, the automatic grading gives instant feedback on returns.

On average, students spent 54 hours on computer exercises, which fits well to the total time allocation of the course. Students found the exercises laborious but educational.

Perception of the workload varied widely based on the background of the students. Intensity of the load could be lightened by improving instructions and by giving some of the designs as partially ready made. Alternatively, a graduated exercise model could be used for the exercises.

Keywords: Digital design, computer exercise, FPGA board, Plussa

TIIVISTELMÄ

Matti Käyrä: Digitaalisuunnittelu-kurssin tietokoneharjoitusten uudistus ja arviointi
Diplomityö
Tampereen yliopisto
Tietotekniikka, Sulautetut Järjestelmät
Toukokuu 2019

Tämä diplomityö liittyy digitaalitekniikan perusteita opettavaan digitaalisuunnitelu-kurssiin. Kurssilla opiskelijat oppivat suunnittelemaan, analysoimaan ja toteuttamaan porttiverkkoja, sekventiaalisia järjestelmiä ja tilakoneita, sekä totetuttamaan näitä FPGA-kehitysalustalle.

Kurssin FPGA kehityslauta on vanhentut ja tietokoneharjoitustehtävien rakenne on sirpaloitunut laajemman toteutettavan projektin ja lisätehtävien välille. Kurssille suunniteltiin tässä työssä kokonaan uusi tietokoneharjoitusprojekti.

Ennen harjoituksien kehittämisen aloittamista kurssin kehitysalusta oli valittu. Tämä oltiin tehty ominaisuusperusteisesti, uusi alusta mahdollistaa monipuolisesti eri kurssien hyvin erilaisia harjoitustyöitä. Harjoitustyön ja alustan uudistamisen yhteydessä otettiin myös käyttöön MOOC-pohjainen verkkototeutus. Se mahdollistaa puoliautomaattisen kurssin palautusten arvioinnin.

Harjoitustyön tavoitteena on sitoa kurssin teoria ja paperitehtävien harjoittamat suunnitteluperiaatteet käytäntöön oikeiden teollisuudessa käytettyjen työkalujen kanssa. Harjoitustyön kohteena on peli toteutettuna kehitysalustan LED-matriisilaajennoksella.

Työssä tehtiin esimerkkitoteutus pelistä ja se pilkottiin viiteen tehtäväkokonaisuuteen: Työkaluihin ja FPGA suunniteluvuohon perehtymiseen, aritmetiikan ja hierarkian suunnitteluun, tilakoneiden suunnitteluun, valmiiden lohkojen laajentamiseen ja integrointiin. Tehtävien palautuksen yhteydessä automaattitarkistusjärjestelmä hakee opiskelijan toteutuksen tehtävästä versionhallinnasta, suorittaa logiikkasimulaattorilla tehtäväkohtaisen testipenkin ja pisteyttää tuloksen perusteella opiskelijan palautuksen.

Simulaatiotulokset pitäisi tulkita manuaalisesti opiskelijoiden kuvankaappauksista ilman automaattitarkistusjärjestelmää. Henkilökunnan kurssiin käyttämä aika voidaan näin ollen tarkentaa enemmän opiskelijoiden hyödyksi. Opiskelijoille automaattitarkistusjärjestelmä antaa välittömän palautteen.

Keskimäärin opiskelijat käyttivät harjoitusten tekemiseen 54 tuntia. Aikamäärä sopii muun kurssin ajan käytön kanssa kurssin kokonaismitoitukseen. Opiskelijat kokivat harjoitukset työläinä, mutta opettavaisena.

Koettu työmäärää vaihteli huomattavasti johtuen opiskelijoiden erilaisista taustoista. Kuormittavuutta voidaan alentaa parantamalla ohjeita ja antamalla osittain valmiiksi tehtyjä lohkoja, tai luomalla kokonaan uusi, porrasteinen malli harjoitusten läpäisyyn.

Avainsanat: Digitaalisuunnittelu, tietokoneharjoitukset, FPGA kehitysalusta, Plussa

PREFACE

This thesis is part of Tampere University, and its predecessor Tampere University of Technology, course upgrades that focus on developing new exercises to replace ones that use old FPGA development board.

I would like to thank both Tampere University and its predecessor for chance to create this thesis as well as project partner whom I created this exercise set with. Additionally, I'd like to thank my coworkers for making the laboratory and the unit after that be great place to work in.

This project, and thesis, started my professional career and inspired me to stay on academic career for now.

Finally I'd like to thank my family for supporting my efforts and being the source of my inspiration to study and finish my thesis.

In Tampere, 9th May 2019

Matti Käyrä

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

EDA	Electronic design automation
D&V	Documentation & Visualization, HDL Designer feature
FPGA	Field programmable gate array
HDL	Hardware description language
ILA	Integrated Logic Analyzer
RTL	Register transfer level
TA	Teaching Assistant
TUT	Tampere university of technology
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuit

1 INTRODUCTION

This Master's thesis explores the project of designing, implementing and evaluating new computer exercises for the digital design course at Tampere University. The course is one of the cornerstone courses for electronics, embedded systems, computer engineering and communication systems studies. It is meant for both Bachelor and Masters studies for both Finnish and international students. The course explores the digital system design process and practical implementation of the created systems using a field programmable gate array (FPGA) development board as a physical target. [20]

The main enabler for the new exercises is the need to replace older development boards. Software support for the EDA tools used with those is expired. The old exercises themselves are fragmented between the main project and supplementary tasks so this opportunity was deemed perfect to update the exercises. Introducing new target board provides clear slate to implement new kind of target project while taking the new board to use at the same time.

The thesis begins with analyzing the course content and the learning flows used. After this, the old exercises are looked into and undesirable features in them are highlighted. Once these aspects are explored, it will be more clear which kind of features will be tried to be incorporated in the new exercises and what kind of things should be avoided.

The thesis will then move on to the process how the new exercise reference project was created and how it was divided to different exercise sets. Each of the exercise set try to teach student major design aspect: Introduction to design flow, arithmetic and hierarchical design, state machine design, extending existing designs and integrating designed blocks. The new MOOC-based exercise web portal, Plussa [23], is also considered as it enables the course to have semi-automated grading of its exercise tasks. After this section, the newly created exercises are compared to the old exercise set and the key differences between them are highlighted.

Latter part of the thesis explores the results from the first implementation the exercise project was part of the digital design course in the fall of 2018. Firstly, in this part, found issues with the new project are explored. Then, time the exercises consumed from both staff and students are looked into. After this, the thesis will have a section where possible solutions for the found issues are looked into and a schedule for implementing the fixes will be suggested. Lastly, the thesis will have a conclusion which summarizes the whole project and its results.

2 COURSE STRUCTURE

In this section, the thesis explores how the course was previously arranged, to understand how things were set up, and why they were as they were, if they are to be improved upon. The course arrangements were studied as they were during the introduction of the previous FPGA development boards. [39]

The course was divided into lectures, paper exercises and computer exercises. In addition to these there is an exam, which denotes the grade of the course. The course has also been possible to pass with grade 1 by completing 75% of the paper exercises and 75% of the computer exercises. This thesis only focuses on the computer exercise part of the course. The most important aspect analyzed is the work flow of the computer exercises and why it is in use. However, it is important to acknowledge other parts of the course too, since they affect how the computer exercises need to be set up.

The learning flow of the course is to explore concepts and ideas on lectures, then completing theoretical exercises with paper exercises before moving on to using electronic design automation (EDA) tools. This ensures that students learn one aspect of the process at a time. The learning process used on digital design course is depicted in the Figure 2.1. [20]

The computer sciences learning process can be depicted with 2-dimensional Bloom's taxonomy [14]. The original version of this was of one dimensional in nature. In computer sciences, and more specifically system design and coding, the process can have more dimensions. This is due to the approach that one can learn things with either theoretical or practical approach. For the digital design course, the used flow of learning conforms to taxonomy's way that there can be a step by step approach to first gain theoretical knowledge and then applying that specific knowledge to practice. If this process would be graphically depicted, theoretical and practical knowledge would be the axes and the process would alternate between theoretical practical steps, trying to reach the opposite end of the matrix.

2.1 Computer Exercise Design Flow

The computer exercises of the course are designed to familiarize students with an actually used design process. This process starts from given specifications and continues all the way to physical verification on FPGA development board. The exercises follow a bottom

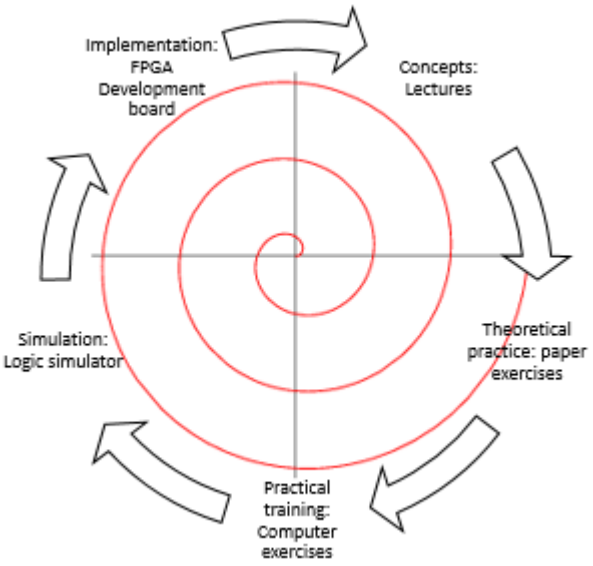

Figure 2.1. *Learning flow of the digital design course [20].*

up approach, as designing a complete system would require more extensive knowledge in a wider range of topics. Focusing on a single design layer helps to focus the learning on a certain topic.

From specifications the system should be unambiguously possible to be designed to conform to desired functionality. While the system can be implemented in variety of ways, it needs produce specified results.

Design flow itself is a iterative process that start with drafting a system and then moves on to logical simulation. These two steps are repeated until the system conforms to the desired logical behavior. Once this is achieved the verification process moves to physical FPGA implementation. This takes the logical simulation results and tries to create a working copy of the system that can be run on development board to verify that it can also work on that platform. Should issues arise at this stage, the process needs to return to design and logical simulation stages. Once design does work on FPGA the verification cycle is considered to be complete and the design can be considered to fill the specifications. This process is depicted in Figure 2.2.

The whole course learning flow and the computer exercise design flow are very similar to Bloom's taxonomy as a spiral. Spiral taxonomy emphasizes that learning process is iterative. This taxonomy is visualized in Figure 2.3. The taxonomy focuses on visualizing learning process to be consisting of various stages that each deepen the knowledge on the subject. It also highlights that it is not enough to simply create designs but also they should be analyzed to gain understanding on the subject. [14].

Digital design flow can be also described with Y-model. Specifically in this model, the design flow is shown with layers of abstractions from the transistor level all the way to the functional level as iterative circles. The process has specific milestones for behavior, structure and physical designs. There is also FPGA specific Y-model. This is depicted in

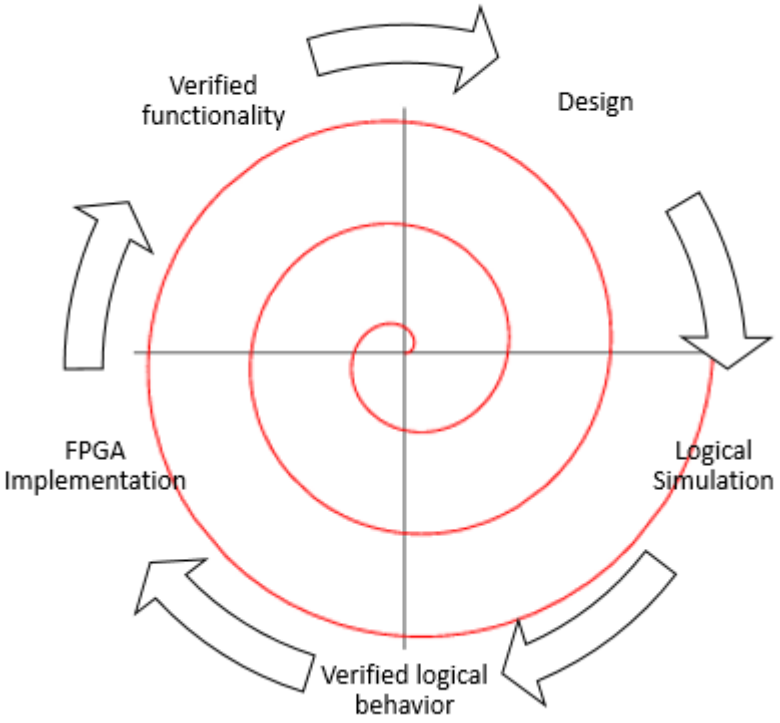

Figure 2.2. *Computer exercise work flow on digital design course.*

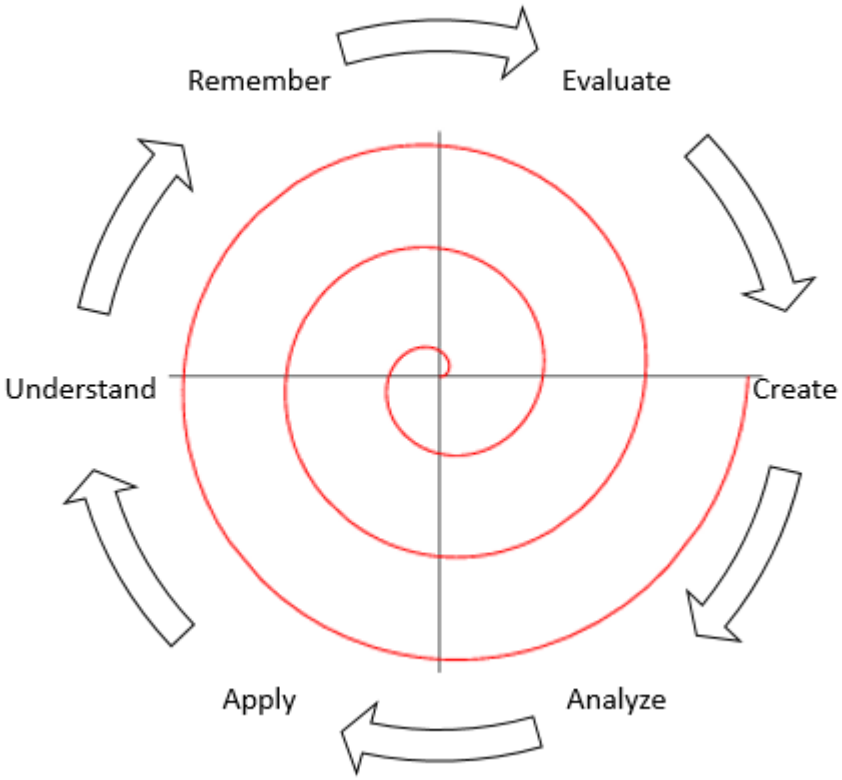

Figure 2.3. *Implementation of Bloom's taxonomy as spiral. [14]*

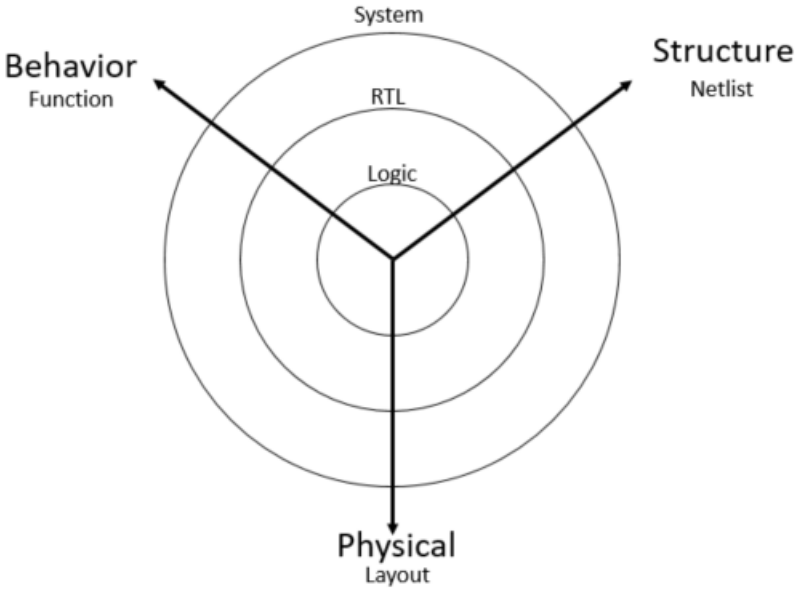

Figure 2.4. *FPGA specific Y-model. [15]*

Figure 2.4. [15]

Course tasks focuses implementing designs on logic and register transfer level (from now on RTL). The computer exercise design process depicted earlier in Figure 2.2 has very similar structure as the Y-model. More specifically, behavior directly the same task as the logical simulation and structure is generated from the RTL made with that step. Physical layout is generated with a synthesis tool, but regardless, it is still in practical terms same step between the course learning model and the Y-model. [15]

Y-modeling can also depict how the design iteration works as both top down and bottom up. On digital design course only bottom up approach is used as it is figured out to be more simple to design with as it is more restrictive. Top down approach would be possible to use but as it is counterpart to bottom up, it can be more complex to start using. [15]

FPGA boards are sometimes considered as only prototyping platforms for chip production, but in some roles, the FPGA is sufficient as an end product implementation platform. This is in particular the case in the products that require or benefit from reprogrammability. From the education perspective, the limitless reprogrammability means that the costs involved in purchasing development boards rarely occur. Earlier reprogrammable logic circuits have been either one-shot programmable or they would decay over time and eventually break down. In comparison, FPGA boards only use time as their spent resource. As all of these are electrical devices, they do use electricity, but that is not considered as a limiting factor. Chip production from designs would have additional tasks but those are not in the scope of this course.

2.2 Implementation of Content

The content of the course implemented in fall 2017 is presented in Table 2.1. The schedule conforms to the designed learning flow presented previously in Figure 2.1. There have been minor issues with the timing of state machine topics, but these have still been explored before they have been needed in the exercises.

In general, the course has had a lot of subjects and techniques to explore. However, it has been very highly regarded in teaching the subject to the students. At the end of the course students have experience with designing techniques and some of the EDA tools used. This should help students to figure out whether the topic is the one that they want to specialize in.

Table 2.1. *Content of the old implementation of the digital design course.*

Topic	Lecture contents	Exercise #	Topic	Exercise contents
Orientation, Specifying combinational logic	Definition of time, basics and specification of combinational logic, gate networks			
Designing combinational logic	Minimizing switching expressions, hierarchical design, decoders, LUTs			
Analyzing and optimizing combinational logic	Analysis, timing analysis, area analysis, power		Familiarizing with exercise materials as self-study	Read the exercise instructions and tutorials on these pages so they are easier to understand during exercises.
The phases and tools of digital design	Y-model, course tools	P1	Designing combinational logic	Simplifying equations, equations from schematics, timing schematics, CMOS, car's cabin light switch
Specifying sequential systems	Time functions, sequences, Mealy and Moore state machines, state transitions	P2	Analyzing combinational logic	Critical path, ripple-carry adder and delays, gate network and delays
Guest lecture	Industry experiences	C1	Tools	HDL designer, gate network and truth table, ModelSim, DE2
Designing state machines	Canonical implementation, implementation example, FSM, designing state machine with HDL designer	C2	Specifying state machines	Debugging elevator logic, 7-segment display, BCD, 2's complement, sequence identifier
Time behavior of FSMs	Timing, state register, signal paths, delay analysis, determining clock frequency, clock skew	P3	State tables and diagrams	ALU design, state table/diagram, state diagram/circuit
Analysis and features of state machines	Analysis, registered, one-hot, HDL designer	C3	Designing state machines	Calculator state machine, transition recognizer, student number sequence, state table/diagram
Arithmetic	Addition of integers, ripple-carry, carry lookahead, and parallel prefix adders, signed numbers and their addition	P4	Designing and analyzing sequential systems	Delay analysis, calculator registers, control logic, sequence identifier
Arithmetic 2, standard modules	Multiplication, shift registers, accumulators, ALU	C4	Designing arithmetic blocks	Calculator's register bank and input block, add/sub/ALU, ALU testbench, door lock with student number sequence
Programmable logic circuits, RTL design	FPGA architecture, history and future, control/data architecture, shared resources, abstraction	P5	Analysis of state machines	One-hot analysis and timing diagram, implementation of functions, pipelining, delay and area analysis
RTL analysis, Multi-module systems	Analysis of data/control paths, data memory, FIFO, buses	C5	Integrating blocks with FPGA	Integrating the calculator, pipelining, delay simulation, keyboard input for chicken game, optional task for FPGA
Recap	Summary, feedback and conclusions of the course			
		P6	Memories and data sheets	Designing FIFO memory, FIFO adapter, using ROM as state machine, reading data sheets

3 ANALYZING THE OLD EXERCISES

The old computer exercise project was to implement simple four operation calculator on Altera DE 2 FPGA development board. The calculator was controlled with switches and buttons on the board with seven segment displays as the display of the calculator. Calculator functionality is depicted in Figure 3.1. Architecture of the system is presented in Figure 3.2. [28, 29]

Development boards were taken to this role when major update on computer engineering courses was done in 2008. After this, courses have had rearrangements, but the digital design course has always kept its place as the first advanced logic implementation course. The calculator has been the main project of the course since the introduction of the development boards. [39]

The project itself was not large or varied enough to be sole content of exercise tasks. This is why it was extended with additional tasks that aimed to increase both variety and choosability of the tasks.

The task descriptions were found on the course website and the returns were done to Moodle portal as compressed folder. Moodle portal also tracks the points given to students, so tracking the progress along the course was simple, however, that was not visible at a glance. Previously, the returns have also been done by attaching a compressed folder to email and Git version control system was tested on one implementation. The information on course has traditionally been scattered across a few different used sites. [16, 19]

The main EDA tools used on the course are Mentor HDL Designer, Mentor ModelSim and Altera Quartus II. These are industry grade design tools which students can potentially use if they start working on the companies that do digital design. EDA tools tend to be relatively similar or at least contain similar functions. Hence, once a student can operate with one design suite, it can be relatively simple process to adapt to use another one. [17, 18, 26]

The design process was limited to use a very strict set of rules, to make both design and verification simpler. Additionally, it would help the checking process as complex components and certain connections would be very difficult to debug. For example, the clock and the reset signals were strictly forbidden to have any components and only register type allowed was a D-type flip flop. While realistically all kinds of various components would be used to make an actual finished product, the reduction of variables that might

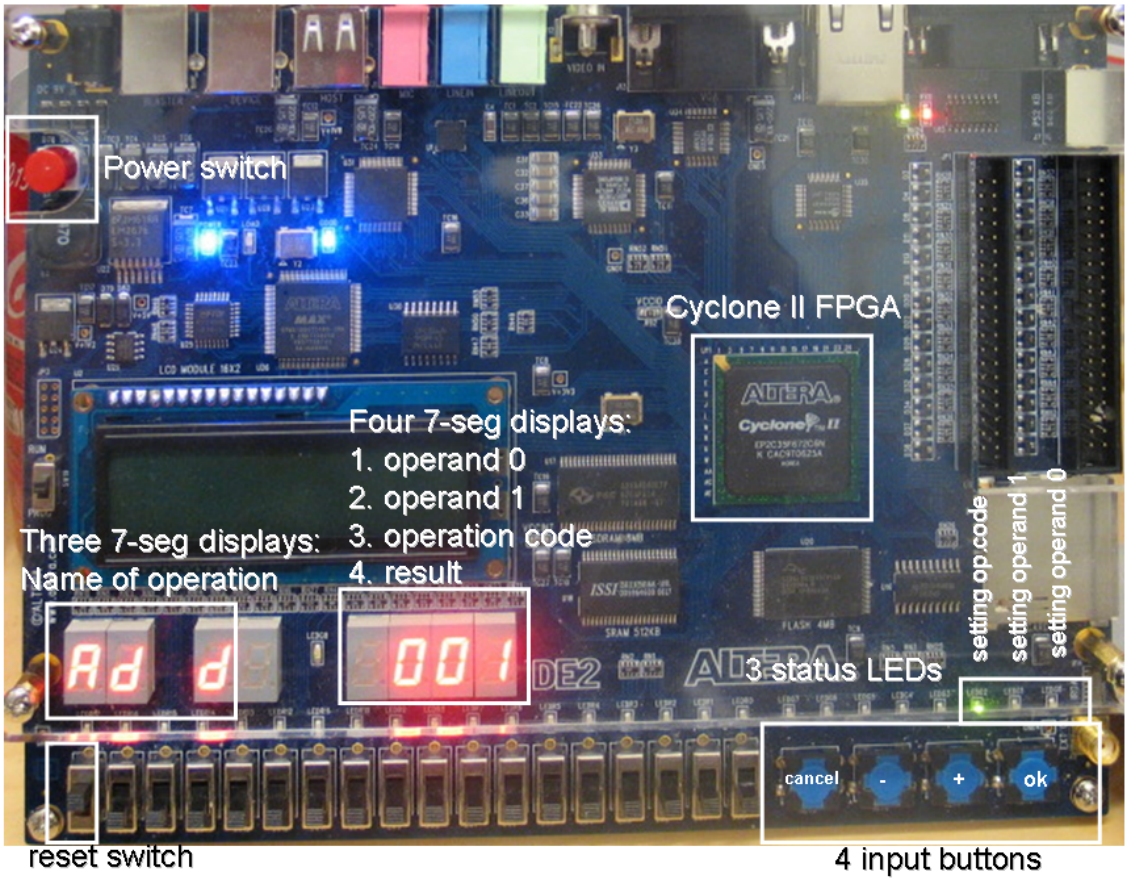

Figure 3.1. *Calculator project implementation on Altera DE2 FPGA board. [29]*

cause issues is the most important factor for strict design rules. This is especially important due to being on one of the initial courses on the topic. [30]

3.1 Tool Specific Design flow

Computer exercises of the course were held in the FPGA class room of TUT. The classroom was fitted with 21 computers with EDA tools and 9 of these were fitted with the course FPGA board. This meant that during guided exercises, students could always focus on the design and simulation parts, but not the FPGA implementation part. Access to this room is restricted. Students, who participate in a course that use the room for its exercises, have access to it. This means that students can also do the tasks independently there.

The old FPGA development board used on course was Altera DE2 with Cyclone II FPGA chip. This chip is the target for logical circuits created with the HDL Designer. ModelSim is the logical functionality hardware simulator that is used to verify the logical functionality. Once the design functions logically, Altera Quartus II is used to synthesize the design and then it is used to program the FPGA chip. Once programmed to the chip, the design functionality is once more verified. This work flow was visualized in generic form in previous

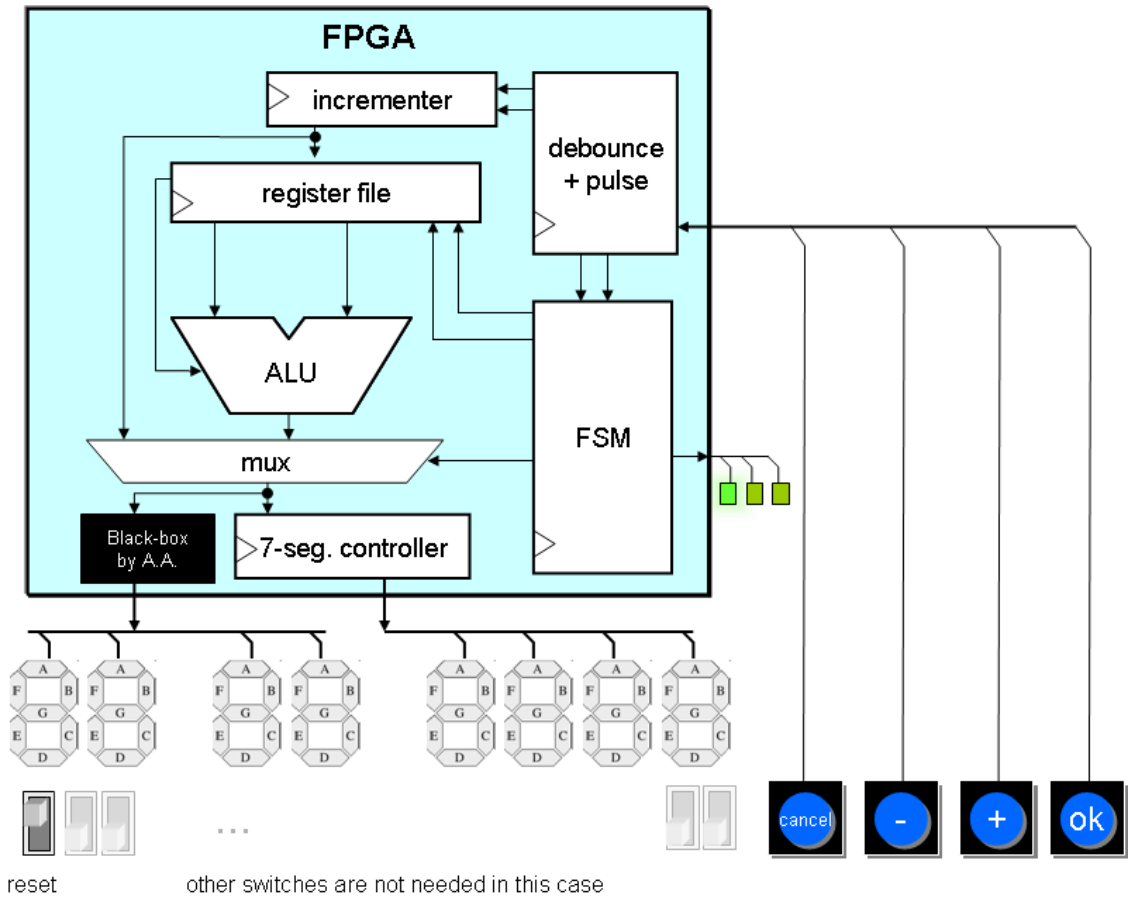

Figure 3.2. *Calculator project architecture. [29]*

chapter, in Figure 2.2. [9, 17, 18, 26]

Additional used EDA tools integrate directly to HDL Designer as tasks, which result in a simple work flow on the computer exercises. The flow consists of designing the logic on HDL Designer diagram, then opening the simulator from HDL Designer, which automatically loads all of the diagrams to ModelSim. The only thing left to do in ModelSim is to actually run the simulation and set the simulation waveform to track interesting signals. In the simplest case, it is one click to run the simulation and then testbench would produce verification output print. This is done to avoid students to be overwhelmed by all of the functions of the EDA tools. [17, 18]

After simulation succeeds, the designs can then be loaded on the FPGA chip after they have been synthesized and routed. These operations are also integrated into HDL Designer tasks. They do have to be run in specific order, which might cause sometimes issues if one of the steps have been skipped. These tasks and the actual programming of the FPGA chip is done by Quartus II tool. Tasks and the programmer tool are opened directly from the HDL Designer. [17, 26]

While all of the software integration eases the work flow greatly, the main reason the HDL Designer was chosen to be the course design platform is the abstraction of the logic with various kinds of diagrams. Once the diagram is ready the tool will generate

hardware description language (HDL) code from the diagram. Most used diagram type is a block diagram, as it allows other diagrams to be combined to create hierarchies. It also allows logical ports and various components to be placed on the layout. This is very similar to electrical design tool schematic diagrams. For electrical engineering major students the tool is relatively familiar, as for an example, one EDA tool used in the courses of the department of electrical engineering is PADS. It is also a product of Mentor, the creator of HDL Designer. Other types of diagrams used on course are state diagrams and truth tables. The software has other diagrams types too, for example such as a data flow diagram, and it also supports writing direct VHDL [5] code, but only the diagrams mentioned before are ones that are meant to be used on this course. [17, 22]

Block diagrams can be used to create combinatory logic diagrams and canocial state machines. However, the state diagrams diagrams are considered as the main advantage of HDL Designer. These are higher level implementation of the state machines. This design type can be taught on paper exercises and the same principles apply to the HDL Designer diagrams. This abstracts the actual working logic which makes making complicated state machines very simple and fast process, comparing to manually coding VHDL for example. This diagram type is explored in detail in one of the HDL Designer tutorials of the course. [17, 31]

When students move to use these diagrams, they need to learn some of the VHDL assignment operations. Moreover, more complex conditions need additional VHDL coding understanding. However, with relatively quick trial and error process, even those that are unfamiliar with VHDL operations, can learn to make use them. Additionally, HDL Designer has VHDL assignment operation help panel, which on click produces VHDL operators to assigned operation spot.

While in practical terms, the HDL code is generally produced with either hand coding or high level synthesis. HDL coding requires the coder to be already familiar with the design process, the potential synthesized logic and logical functionality of the created logic. If it is tried to combine all of the aspects of these, result would be far too complex and time consuming for them to be one single course. Time allocation of single course is approximately 125 hours. In Tampere University this is solved by separating VHDL coding and practicalities of the synthesis to separate course, logic synthesis. This allows digital design course be purely of the designing logic, simple verification and general work flow. [21]

The main tool, HDL Designer, like most of the EDA tools, is quite precise how it must be used. The greatest advantage is that it allows producing the HDL code without actually coding HDL code. This saves the learning effort to the actual design and not learning the coding syntax. General consensus has been that once enough practice has been done with the software, it will not cause any issues. In terms of exercise sets, enough software training has been done by the second set of exercises.

After all of the exercises for each of the set were completed, the documentation & visual-

ization (From now on D&V) functionality of HDL Designer is used to produce a web page that contains all of the student's designs. Also, simulation waveform screen shots were required from all but purely demonstration tasks. If either screenshots or D&V were found not sufficient, the student would be given chance to fix the issue. All of the various needs to fix diagrams were required to be able to pass the computer exercise part of the course. [30]

Students were also allowed to loan the FPGA board to use with their own computer setup. Course provided instructions how to install and use the EDA tools. These were quite extensive, but each year, some of the students did loan board. EDA tools required licenses and these were available through tunneling. Demonstration tasks were still required to be done in the classroom. [31]

3.2 First exercise set

The purpose of the first set was to introduce students to the development environment. First, before starting, HDL Designer libraries needed to be set up. The tasks themselves were first implementing a simple half adder and then adder component. The adder component was to be implemented in various ways. Both of the designs were simulated using ModelSim and lastly half adder design was implemented on FPGA board. All of the tasks in this set were mandatory to do but they were not directly part of the exercise project. [32]

After this set the students were to understand how the programs behaved and how to both simulate and to implement design on hardware. Detailed tutorials for all of these tasks tried to ensure easy introduction to the EDA tools. As FPGA prototyping was introduced, students also gained practical feel out of the tasks. [31]

3.3 Second exercise set

The second exercise set consisted of a debugging task, seven-segment interface module, seven-segment controller and lastly simple sequence recognizer module. While task descriptions claimed that both of seven segment modules were part of the calculator project, only the latter one actually was. Debugging task and seven segment modules had FPGA part in addition to simulation. [33]

The debugging task was about identifying and fixing bugs in a complex elevator simulator. It was hierarchical design with various modules. Students were to debug the elevator driving logic with the help of a simulator tool and fully functional FPGA implementation. Once students have fixed their design, they will use their own version on FPGA to see that it really works. FPGA implementation has visual representations of the logic with LEDs and floor number. [10]

Seven segment modules are similar to each other, but only second one was the display controller for the project. With them, numbers would be produced to seven segment displays on the development board. The first one would take Binary coded decimal input and produce a control signal from that. The latter would create the control signal from traditional 2's complement 8 bit binary input. For this, students were to implement a binary to binary-coded-decimal conversion algorithm with given Add-3 module. Module is based on Verilog module that was distributed publicly in the University of Dayton website [41]. Both of the modules have their own FPGA design that was to be implemented on DE 2 board. [33]

Last task of the set, the sequence recognizer, was not connected to the main project at all. It was a supplemental task to increment the number of sets tasks to four. It helped taught how serial signal can be used in addition with registers to create a control sequence. This concept can be useful in digital systems but it is not utilized in the exercise project. This task did not have a special FPGA platform to implement. [33]

Due to heavy wiring and bit accurate algorithm implementation, seven segment task was highly time consuming to do. Also, finding bugs with very limited previous experience is very challenging. However, since there are multiple implementation tasks to do, the set feels very hands on practical. Additionally, as there are multiple options to choose from, the trickier tasks can be skipped if the student chooses accordingly.

3.4 Third exercise set

The third set of exercises introduced new HDL Designer functionality, state diagrams. These are used to implement a finite state machine (from now on FSM) designs with abstracted implementation. This design flow relieves the designer from having to design the system on register level. Tasks explored delay with FSM design and in addition to that, there were three tasks about Mealy and Moore type FSM implementation. While the delay simulation was done to FSM controlling the calculator project, it was a version that was not directly included in the main project. Other design tasks were to implement button press pulsifier, a sequence generator and a simple state machine according to a given state table. [34]

Pulsifier and sequence generator had FPGA implementation part. Also, pulsifier task had additional part that tasked students to take a look at created VHDL code. This point was likely the first time that student would see actual VHDL code despite generating it with HDL Designer from the very first exercise. [34]

With this exercise set the student has now learned all of the various HDL Designer functions used on course. Arguably the most powerful tool, the state diagram, should now be familiar to the students. While this thesis only focuses on the computer exercise part, these diagrams can be designed on paper to a great extent and then simply redrawn to the HDL Designer. As with previous tasks, some of the tasks of this set, namely the state

table task and sequence generator, are not part of the main project and they really feel disconnected from it. Additionally, the ordering of the tasks suggests that the state table task is simply supplemental to create equal sized exercise sets. In total, this exercise set is less time consuming than previous, which might feel odd to the students. [34]

3.5 Fourth Exercise set

In the fourth set of exercises students implement register bank and its controller, ALU (Arithmetic logic unit), testbench and sequence recognizer. Register bank, the controller and ALU are part of the greater course project. The only synthesized design of this set is the sequence recognizer task. This makes the set very simulation heavy compared with previous sets which have larger part of implementing the designs on FPGA board. [35]

The register bank contains the operands and result of the calculator. The bank controller determines where the data is being saved. ALU provides the various functions that the calculator can do. The testbench task is not connected directly to the exercise project, but as it is used to confirm one of the operations of the calculator ALU, it can be considered being connected. This task also encourages students to familiarize themselves more with VHDL code. However the testbench itself still should be implemented with schematic diagrams. [35]

The last task of the set, the sequence recognizer, is not part of the exercise project. However, as this is not the first such module, it has the feeling of progressing to more complex designs. It can be thought to be a variant of the sequencer module created on the exercise set two. As such, it has its own connection within the set without being connected with the main calculator project. In comparison to other tasks of the set, it still feels little more than filler exercise. [35]

At the end of this exercise set the student should be more familiar with registers and hierarchical designs. Additionally, while being optional, understanding testbench design is a very important concept in the design process. Whether being designer or verification engineer, testbenches are very powerful tools.

3.6 Fifth Exercise set

Tasks in the last exercise set consisted of assembling the whole project, pipelining and delay simulation, integration and additional choosable exercise. Choosable task options were implementing missing project components, button debouncer, a LIFO or a FIFO memory buffer, arbiter or combinatory logic gate design. This set of exercises has a large collection of fairly complex systems to augment the design and testing process. [36]

Pipelining and delay, integration, memory buffer and button debouncer tasks all have rather elaborated additional ready made files to test the design with. The pipelining and

delay task has a custom created components which have delay inserted into them in addition to the functionality. The delay within these components is created by VHDL wait statements. This makes them simulate the actual behaviour of the logic gates more accurately, but it makes them not being able to be synthesized. The memory buffer task has custom seven segment interface to enable testing it on the FPGA board.The button debouncer task has an oscillation generator on both simulation and FPGA parts of the task. [36]

The integration task of this exercise set is to implement a Chicken game, two-dimensional platform game, a design to be run on the FPGA. The game is provided as HDL files which are then connected using the Quartus II integration tools. This gives student insight how various intellectual property (from now on IP) blocks can be connected without creating connections on code level in HDL Designer. [6]

At the end of this set, students have been familiarized with EDA tools and the design process used on course. This process emulates the real design prototyping process. Depending on their choices they might or might not have assembled the core project of the calculator. Regardless of this, the students who pass the exercises have the basic knowledge of how to implement digital systems.

3.7 Issues with the old project

The exercise project initially consisted of the calculator project and later additional exercises were developed for it. These have extended the options on each of the weeks, but at the same time, it has created a feeling of exercises being unconnected to the actual greater project.

The aging development boards have also been somewhat of a problem by themselves, as due to age the mechanical switches have started to be unresponsive due to mechanical wear. Other issue is the age of the required software. The course exercises use a classroom with Windows 10 computers. However, some of the EDA tool versions do not have official support with the used operating system. This means that if there is software issues, those can be very difficult to debug, especially if it is something that is caused by the incompatibility. Additionally, the web drive system used at TUT might also contribute to issues if the used EDA tools are not up to date and designed to work with various types of setups.

Due to information being scattered through multiple portals, the course website and Moodle, the information seemed like it was not always readily available. Additionally, navigating to the correct tutorial page for example seemed too tricky for the gained information. The availability of the information pages is very important for the students as the information on the EDA tools are not readily available elsewhere, as they are not free programs. The tutorial pages also contained an additional irrelevant page which was not used by the course. [31]

Instructions on tasks were bare minimum required to complete the tasks. This was usually enough but in many cases students have been somewhat lost where to start and how to create the design. These were partially a design choice, as stating all of the information usually is unnecessary. This is due to the provided testbench which exposes all of the ports required. From these students get a good starting point and should be aware of most of the details from the specification of the task.

As all of the files for the project was distributed as the exercises begun, the student would feel overwhelmed by the number of designs in their HDL Designer course library. While all of the design top levels where students connect their own designs are named in the exercises, it can be confusing to find the modules, as their names are marked with "_tb" suffix. Due to designs being ordered by name, this leads to various testbenches being scattered all over the file list regardless how one tried to sort it.

4 IMPLEMENTATION OF NEW PROJECT

This chapter explores the new target platform for the computer exercises and the design flow used with it. Then it moves on to creation and partitioning of the new exercise project. Lastly the chapter also covers the MOOC-portal taken to use with the course.

When the project was taking place development boards of a multiple kind were currently used on the computer engineering courses of TUT. Standardizing on single board was desirable. It was found out that the old design process could be updated to PYNQ-Z1 boards, that could also be used to other courses as well. [24]

The general course structure has been kept main unchanged. The learning process on course has not been altered, the first concepts are taught on lectures and then used on exercises. Exercises are still divided into both paper exercises and computer exercises. The traditional exam was given an electronic alternative, which is the TUT EXAM examination system. Similarly than before, the automatic passing with grade 1 still exists, but instead of requiring the specific percentage completion of both exercise categories, simply 75% of all of the exercises is required. This means that students can more freely decide for themselves to which part of the course they focus if they can not try to learn all of it. Bonus points to the passed exam are gained linearly after gaining half of the available exercises.

4.1 Target Board

The PYNQ-Z1 board was chosen as the new development board used by the digital design course. As the board provides a platform for both general programming, as on the introduction to programming course, and digital logic, it was a natural candidate that to standardize on. The board was chosen before the actual development of this exercise project begun.

The board is Digilent product that uses Xilinx Artix-7 family FPGA chip xcz7020clg400-1. It has plenty of FPGA performance capability and it also has architecture that is suitable for more complex projects. This course uses only the programmable FPGA logic, but there is also ARM cortex A9 processor on the chip. [24, 42]

Additionally, the development board has plenty of various ports for extending the platform. The main extension used by this course is the Arduino shield pin system. It is used to

have ColoursShield constant current LED Driver and 8x8 LED Matrix in connection with the board. [4, 7, 24]

Arduino modules are generic ones and they have plenty of manufacturers, which all produce their own variant of the module. These all conform to the same Arduino layout and they can be driven with the same code on Arduino board, or logic in our case. [4]

In addition to digital design, also other courses on the laboratory are at the same time being converted to using this board. Previously the next course in the digital technology specialization, logic synthesis, used the same Altera DE2 than digital design. That course was also converted to PYNQ-Z1 board at the same time as digital design was. Logic synthesis, like digital design, creates designs that are synthesized and then programmed to FPGA chip. [21]

In addition to courses mentioned, other courses that were using the other laboratory development board, containing Cyclone V chip, were converted or in the process of being converted to use this board instead.

4.2 New Exercise Design Flow

The work flow with the new platform will be similar to the previous implementations. HDL Designer is used to produce the design diagrams and then integrated ModelSim task is used to simulate the design. Since the FPGA development board has been changed, it is required to use new synthesis and programming software, Xilinx Vivado. [40]

Similar to HDL Designer block diagram tool, Vivado also has its own diagram drawing tool. This, however is meant to be more like IP integration tool and not directly logic design tool. Diagrams there could be used to include chip specific logic blocks or connections between various components. These were deemed to be avoided on this course as it would force the need to learn even more of the various tools and this would draw the focus from the design flow itself.

Vivado does not directly integrate into current version of HDL Designer used on the course, as was the case with Quartus II. As such, work flow has been altered to import the hardware design files produced by HDL Designer, to Vivado, which then can be used to synthesize and program the logic to the FPGA chip. This has been deemed more general approach to FPGA prototyping as it does not require the user to have HDL files to be generated by a certain program.

It was hypothesized that it would be possible to provide students with script to fast set up a Vivado project. This, however, was not implemented as it would likely cause confusion as the script has to be run in a certain way and it only allows unique named projects to be created. This would mean editing the file after each creation. It was additionally figured out that the time it took to create single new project that it was not worth the potential issues that would have come with this approach.

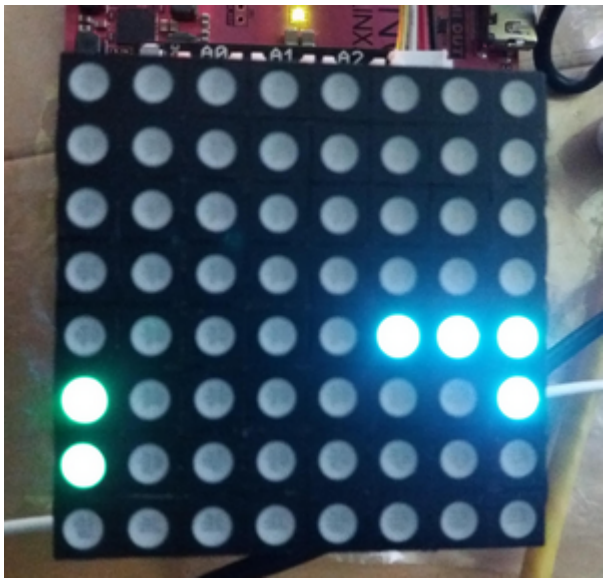

Figure 4.1. *Example Python implementation of shooter game on PYNQ-Z1. [1]*

4.3 Target Application

The new exercise project was inspired by the exercise of the introduction to programming course spring 2018. That exercise resulted in a shooter game, which used the PYNQ-Z1 development board and Arduino constant current LED driver and 8x8 LED matrix. It was about timing a button press right, which would result in a hit on a moving pixel. In that course, the project is implemented in Python programming language and it uses the Jupyter notepad system and relatively complex driver system to drive the led matrix. The functionality of the led driver system is abstracted in a way that it only requires the Python code to input colour and coordinate to light up led. An example of visuals of the game is presented in Figure 4.1. [1, 4, 7, 24]

A simple version of the game is relatively simple to code for programming students. A pseudocode example of implementation is shown in Program 4.1. The program loops movement for the target and button press causes check whether the target was hit. This version of the game stops when the target is shot. However, variations to the game can be made and improvements could be easily added.

The game was selected as the target system because it was deemed modern and interesting. If students had done the Python exercise earlier, completing the exercises would give them additional perspective that the same target system can be produced with various means. Even if they did not participate on that precise exercise but did use the platform for other exercises or attended other implementation it would help to highlight the difference how long designing and implementation takes between hardware and software.

```
1  def game():
2   while True:
3    if button 1:
4      end
5    for i in range:
6      led_matrix.draw_new_target(coordinate)
7      wait_for_update()
8      if button 1:
9       led_matrix.draw_end()
10      end
11   led_matrix.hide_old_target()
12    if button 1:
13     end
14    for k in range:
15     led_matrix.draw_new_target(coordinate)
16     wait_for_update()
17     if button 1:
18      led_matrix.draw_end()
19      end
20     led_matrix.hide_old_target()
21
22 def main():
23  init()
24  while True:
25   led_matrix.clear()
26   led_matrix.draw(player)
27   game()
28   wait_at_the_end()
29
30 main()
```

Program 4.1. *Pseudocode implementation of shooter game.*

4.4 Implementation Process

The project begun by implementing the game with Python, to understand what would the desired system look like on the PYNQ-Z1 board fitted with the LED matrix. Program 4.1 was the result of this effort. After the python coding, the system needed to be implemented in more precise coding, as the Python implementation would abstract too much of the details out. There is plenty of ready made Arduino library materials to use to drive the shield, and implementing those into pure C code was the next step. [4]

The C code was programmed to Arduino Uno board to verify its functionality. Through some effort, the C implementation worked on board fitted with the same LED matrix and the constant current driver. Though, at this point, it was feeding random colour data, not actual game. Once the code did work on Arduino, it was figured that it should be ported to Linux running on the PYNQ-Z1 board. This porting was not successful, and after some time it was abandoned. Instead it was simply tested that Linux I/O ports could

be activated with a C program or direct commands. These ports did not translate back to actual usable port identifications, and it was figured out that it would not be beneficial to try to figure out more with that. [4]

What was concentrated then was the C implementation on Arduino. It was looked into how it actually functioned and what were the actual pins that were driven by it. The pin layout of the PYNQ-Z1 board had to be looked in to, as it was known that the FPGA or the processor could simply drive the Arduino header pins on the board. It was found out that the pins can be driven straight from the FPGA part, without needing to use the processor to anything. For digital design course purposes it was the perfect solution, as the processor could be simply ignored. Everything about the design could focus on the FPGA. Implementation platform resembled greatly previous Altera development board as it did only have FPGA and not processor. [24]

Vivado suite uses constraint files to denote rules for the synthesis and the implementation, such as previously mentioned pins. A general purpose constraint file for the course was created, which contained all of the various Arduino pins, buttons, switches and LEDs. The board also supports running 5 different FPGA clocks, and one of them can be set through the constraint file. This in practice meant that there was no need for complex operations in Vivado, which would simplify the usage on course. Less used functions generally means less problems for the new users, so it was deemed a beneficial factor for the upcoming exercises. Few mock up designs were created to verify that the correct Arduino pins were functioning with desired voltage, as the PYNQ pins can be driven with either 1.8 or 3.3 V. Voltages were measured with a traditional voltage meter. Restrictions to these voltages did not affect the LED shields in any way, but it might need to be considered if the shields had strict restrictions within their specifications. Additionally, Vivado requires the voltage level to be set other than the default, othervice it will cause errors in the implementation phase. This can be set in the constraint file, which would mean that if the correct file is used, there should be very few operations in Vivado, which should later translate to very few issues with using the program. [40]

At this point, there was a clear implementation system and it was clear what kind of functionality was required in logic to produce something on the LED matrix. Shield controller design started with the implementation of a shield reset module. After this, a transmitter module for colour data was the next step. This was followed by a gamma value setting module. The gamma values of the shield are responsible how bright are each of the LED colours. With addition of a module, that looped the row activation signals, what was combined, was a system that could produce something visible on the matrix. At this point, it was just a proof of a concept system: no actual data could be produced to the LED matrix at this point. [7, 11]

The next step in the system was implementing a stack of registers that would hold the actual picture data that would be sent to the shield at the time. Since the LED controller needed its values to be constantly transferred even with a still image, initial design was improved to have two stacks of registers. This would result in a system that would only

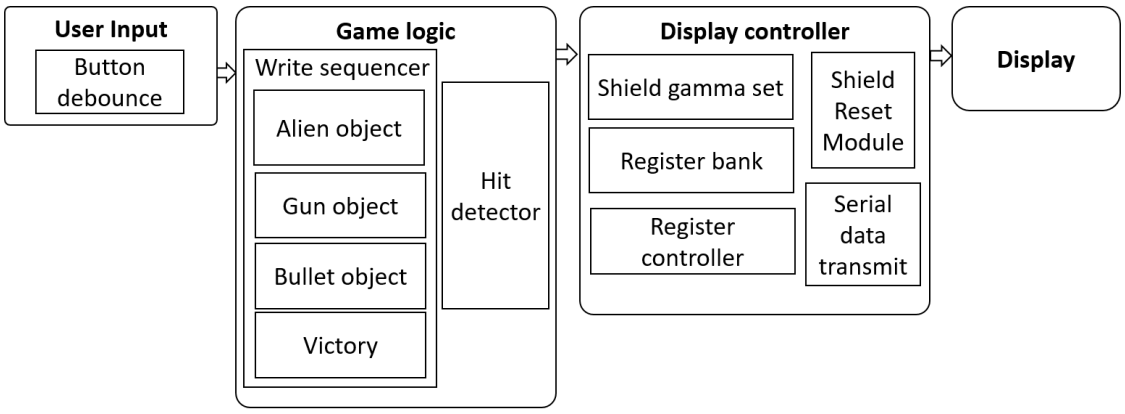

Figure 4.2. *Architecture of alien shooter game.*

transmit the data of full images to the shield. This would mean that user could take as long as it was desired to make updates to the next image. Last addition to the display controller module is a memory controller to be an intermediary block between registers and the serial transmitter. This module would now also have the responsibility of triggering the control signals to the shield, such as the row activation channel signal and latch signal. Latter is transmitted after each row of data to set the values to be shown. [7, 11]

With display modules in place, implementing various game logic functions was more straight forward process, as the display controller will take care of all of the functions that are relevant to driving the LED matrix correctly. The abstraction layer between the game logic and the display controller consist of 3 control signals for writing the data to registers, pixel register address and pixel colour data. The control signals were invented based on the need on the spot without prior knowledge of common handshaking methods. They correspond nearly exactly to basic ready to receive - transmit done pair. In addition to those two signals, design includes a write signal which is used to write the transmitted data to the register bank. If the write signal is not active, the incoming data is discarded. As the control signals correspond very close to basic handshaking, in theory they could be tried to be used with various templates of handshaking protocols, but as of this thesis creation this has not been done.

The game logic is controlled by a write sequencer module which gives writing turns to each of the modules to transmit their coordinates to the display controller. It also takes care of controlling the update speed, and lastly it controls handshaking signals with the display controller. Game functions were divided into modules based on their role, such as a movable gun platform, an "alien", a hit detector and an end effect. Architecture of the game is shown in Figure 4.2. These were later deemed to be part of the basic version of the game. In Figure 4.3 the running game is shown. The reference design was also tweaked to include a few improvements: alien was given additional required hits to win the game and there was a speed switch.

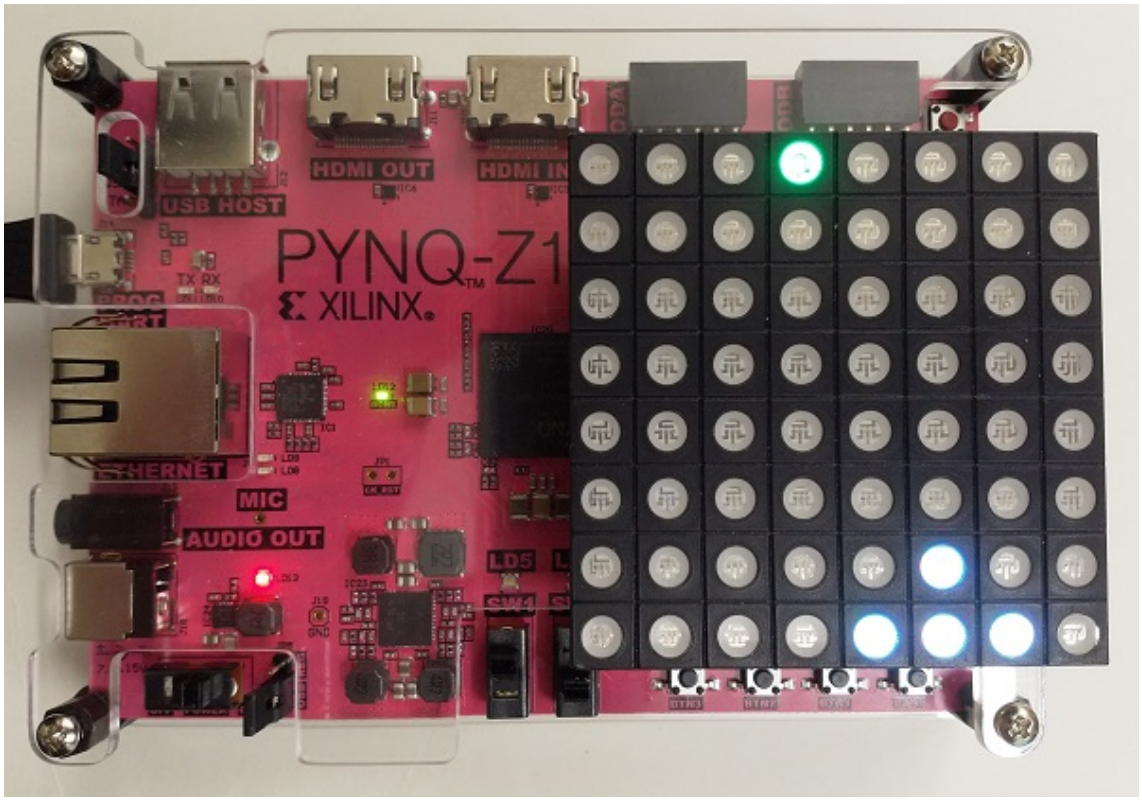

Figure 4.3. *Complete reference project running on PYNQ-Z1 board.*

4.5 Task Design Process

The project at the first glance would be very though challenge, so it was figured out that bottom-up approach, as with the previous project, would be replicated for the project. This approach lets the student concentrate on implementing the logical details, and relieve them of the larger system design. Logic design was modular enough that it was easy to denote each to the blocks to a single exercise task. Division of the modules is depicted in Figure 4.4. The tasks first focuses on one of the modules and then move on to rapidly implementing various needed modules. One of the ideas that were tried to implement, was creating the same functionality in a variety of ways, to give perspective to being able to create the same functionality in multiple of ways.

One of the challenges that arose, was that how would the students be able to gain the use of the LED Matrix, and really gain good hands-on feedback from the FPGA with still managing to let them create each of the logical component. To this purpose, obfuscation script was created. It took the VHDL code that HDL Designer produced to the reference project, and it calculated hash values to replace all of the variable names. This scrambled the code sufficiently, that it was practically not viable to try to decode. While the students participating in this course are likely to never looked at the code, the possibility is certainly there, that they would know it and instead of designing the logic, they would simply read it off the code. This felt like real potential issue as the continuation course to digital design, logic synthesis, teaches students how to write VHDL code, and its implementation

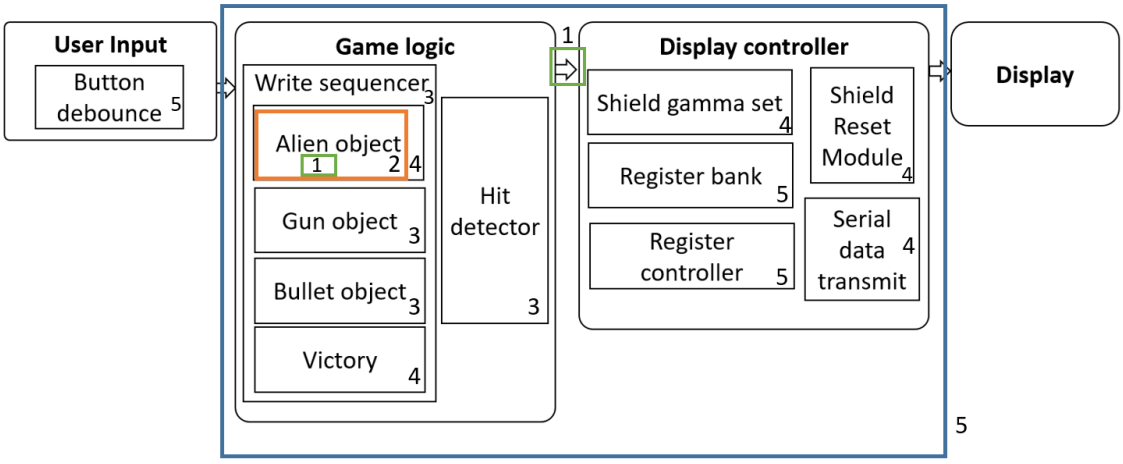

Figure 4.4. *Architecture modules marked with their implementation exercise.*

overlaps the latter part of Digital Design course. [21]

With the black box display controller, the students could initially focus on producing the game logic parts. Once the parts would be ready and tested, they could synthesize the part with the black box display controller to get the led matrix light up and get the practical hands-on feel right away.

Nearly all of the blocks would be created with the help of self-asserting testbenches, to confirm the functionality of the designs. In a few tasks, it was deemed more appropriate to have testbench that does not confirm functionality. These were tasks where the demonstration of the system was the return. It was also deemed beneficial to teach how to manually drive the simulation, to enable students to more elaborately test their designs without ready given testbenches.

Before starting with the computer exercises, the students were to successfully solve a questionnaire about the most common problems and questions that have been arising with the computer exercises. These were meant to ease the staff workload of repeatedly answering the same basic questions. The questionnaire was deemed to be a mandatory task that has to be completed to as part of the exercises.

Task subjects would range from designing an arithmetic component to more complex designs. These were meant to be implemented with the schematic tools of HDL Designer and later to move on with the control style logic to state machines and state diagrams.

Computer exercises are, as with previous implementation, still divided in five sets. The first and beginning of the second set were created to be introductory tasks that would help to learn how to use the EDA tool with their relatively easy logical design. It was thought that students would not have much trouble with the introductory designs themselves and the help of course staff would be needed to help learn how to use the EDA tools. Minimum passable exercise score was slightly over half of all of the exercises. This score results in the fully functional basic implementation of the computer exercise project, resulting in

Table 4.1. *Major topics of the new exercises.*

Exercise set	Main Topic
–	Questionnaire about computer exercises
1	Introduction to Tools and Interfaces
2	Hierarchical design
3	State Machines
4	Design extension and Data sheets
5	Integration

fully playable simple alien shooter game.

As the design process was the same than with the previous implementation, it was decided to reuse the strict design rules that were earlier. These were deemed necessary to simplify the designs and make the students avoid using more complex functions that would be harder to debug. [31]

The Git version control system and the Gitlab repository system were integrated into the course. These were included to ease file management and to manage the exercise returns. The repository was also used to provide the course given files for the project. [16, 38]

4.6 Contents of the Exercises

The main content of each of the new exercise set is presented in the Table 4.1. Each of the major topics was tried to be emphasized in their exercises, but they were not meant to be the only topic in them. Questionnaire about the exercises was meant to be solved before the student even tries to open any of the tools. It had in it questions about the subjects presented in tutorial pages of the course.

As general overview, exercises try to cover most of design aspects explored by the course lectures. Tasks themselves were drafted based on the idea of creating the whole project rather than focusing on trying to implement all aspects of the course. Specific task descriptions are explored in their own subsections.

4.6.1 First Exercise Set

Before starting with the exercises, HDL Designer required three operations to be done to it. Firstly, the settings have to be set, secondly the project has to be created and thirdly ready made libraries have to be added to the project. All these operations were explained with detailed tutorial pages. The same steps were required in the earlier project with the exception of the Git repository being the target folder of the project files.

The first set itself was to be an introductory set would repeat the half adder task from the previous computer exercises and slightly extend it with precise instructions regarding the FPGA implementation. This would introduce the Integrated Logic Analyzer (from now on ILA), the integrated oscilloscope of the FPGA chip, to analyze how the logic functions within the chip. An additional module was created to time the ILA in a way that it can be run with the system clock of 50 MHz in a way that it is perceivable to the student. Additionally, there were two short tasks to introduce students to use the black box display controller, first without inputs and then with interaction.

All tasks of the first set were deemed mandatory. After completing the first set, students would have been introduced to design tools and the working flow with them. They would also have some practice with using the interface of the given display controller.

4.6.2 Second Exercise Set

The second set was about implementing the desired function multiple ways to create identical output. The target for this was deemed to be the movement of the alien. This would, in practical terms, be pixel that would go back and forth on the edge of the LED Matrix. The movement system was to be implemented with either arithmetic addition and subtraction with binary to one hot conversion, or shifters. This tried to highlight that within digital logic there are multiple ways to reach the same end product. This would repeat the idea on which the project was based, reaching the same result can be done in various ways.

In HDL Designer, there is a component library, moduleware, that contains various types of arithmetic components, such as adders, but these were meant to be created from the scratch with logical gates, or from the half adder created earlier. Once a pair of these modules were created, the task was to implement the whole alien module. This would consist of turning logic, coordinate incrementer and coordinate decrementer. The system would also need to recognize button press as initial implementation of the hit logic design. Additionally, the system needed to have register to save the coordinate. The set also had debug task which could be used as hint how to implement the alien module.

The black box display controller was also augmented to give the student an enable signal to their design to increment the coordinate. This signal was later used to modify the run speed of the game. The alien task of second set was a notable increase in difficulty, as it introduced the D-type flip flop register component and the rules to design with it.

4.6.3 Third Exercise Set

The third set of exercises focused on creating additional game modules. These included a write sequencer component, a gun module and a hit detection module. An additional

voluntary task on the set was about creating bullet flight modeling to the game.

The first task was to create the gun module implementation, which would produce a player ship object as a playable character on the matrix. This was meant to consist of multiple pixels in shape that was up to the designer. Second task was likely the most important task of the set in which students were to create a module which would cycle through all of the pixels created by the game.

The write sequencer would later be modified to make it recognize the end of the game and display additional pixels if necessary. The third task of the week would be creating a hit detecting module and this would result in basic playable game. These three tasks were marked as mandatory. The fourth task of the week is creating bullet flight animation to augment the game. Write sequencer and hit detection modules should also be created to include this improvement or modified upon creating the flight module.

4.6.4 Fourth Exercise Set

The fourth set of the exercises was about finalizing the basic game modules by creating an end effect for winning the game. Other tasks of the week were deemed to be choosable. These tasks were improving the alien game and creating first parts for the display controller. These display module parts were a reset module, a gamma value module and a serial transmitter module. All of these three modules had the idea of reading data sheets to produce the required functionality. These were meant to be created with the state machine diagrams of the HDL Designer.

As the display modules cannot be implemented into the FPGA board without all the other parts, it was deemed that testbench would be enough at this point to verify the functionality. The FPGA implementation would be assembled as a task of the last set. In theory, as all of the modules had their own testbench, passing testing with it would ensure the functionality of the assembled system. However, the exercise project was designed to be passable without implementing any modules into the display controller.

4.6.5 Fifth Exercise Set

The last set consisted of creating the two last components to the display controller, a register bank system and a register to serial data component. Additonally, the set had an input debouncer design task and lastly task about assembling the whole project. Complexity of this step depended on the number of the display control modules that were produced. The premade black box display controller was also refined to have proper port names to enable easier assembly on every hierarchical level. The logic within the block was still obfuscated.

This set was also designed to be passable with just the assembly and the debouncer,

without the need to create the display logic. While the register bank was fairly complex design, it was divided into smaller sections. These being a register cell and a combining stages. This was meant to ensure that proper hierarchy was used, and the student did not use too long on drawing the same logical functionality over and over.

The assembly task had simulation part and ILA part on top of the general assembling. It was meant to create the last impression verification chain to be always of the simulation before FPGA implementation. While the various controller interfaces were explicitly defined, the assembly was deemed straightforward to create and no precise instructions were created to the task itself. Also, the roles of each of the modules were defined in their own tasks.

The debouncer task, its testbench and FPGA oscillation circuit were reused from the previous exercise project. It was figured that debouncing the inputs was a good thing to do, even in a project like this. In the DE2 exercise task was mention how the old board has good oscillation dampening and the debouncer would not be needed in the designs, and the PYNQ-Z1 board seems to be as good in this regard.

4.7 Design Verification

The verification process does not fundamentally differ from the previous implementation of the course project. It starts by the simulation verification of the logic and moves on the FPGA board to confirm that verified logical designs do work on live FPGA circuit.

The design verification is integral part of the design process. It is also very large part of the actual exercise tasks themselves. Most of the verification is done by testing is with simulation in ModelSim. Testbenches on course are provided ready made as creating them requires more extensive knowledge of actual coding, which is not the topic for this course. Also, most of the additional modules created for the game project will be required to be tested on FPGA board to verify their functionality when combined with the rest of the project.

A design testbench is a design that provides both stimulus to the system in verification and monitor that verifies whether the output matches the desired function. Responsibility of testbench is to ensure that design both fulfills the higher specification and that the design behaves in a specific deterministic way. Verification in this course is simplistic enough that it does not use coverage analysis or invalid data feed methods. In product environment, these would be used to make sure the product behaves in a deterministic way in every possible case. [15]

Creating testbenches for the tasks was one of the major time consuming processes during the project. Testbenches were created to include both assertions to confirm the correct functionality as well as notes and errors to monitor events in the simulation. These notes and error messages were meant to pinpoint interesting times to look at the wave-

form. Additonally, the designs were to have time out conditions programmed to them, to end the simulation if it was clear it was not going to pass the test. This was mostly due to automatic return environment but it should help make the student sure about the need to improve their design.

Testbenches that were created for the project varied in the implementation style. There are both fully clocked hierarchical designs and timed stimulus testbenches. Furthermore, there are golden design tests and data comparison ones. All of these can be explored by the student to give ideas how to create designs. Testbenches are mostly hierarchical systems created with writing direct VHDL code, however the hierarchy itself and state machine diagrams are created with HLD Designer diagram tools. This gives the curious student chance to inspect the inner workings of each of the testbench to understand more how the testing platform works. This, in turn, can teach useful things about the design process. However, all of this is considered optional extra and the student should be able to progress on course without this.

FPGA verification is very straightforward continuation to testing with testbench, with the exception that it requires an additional tool for synthesizing the design as FPGA programming file. The platform also enables additional debugging with the ILA functionality of the Vivado and FPGA chip. Other than the ILA waveform, the tested system should be stimulated with inputs to verify its functionality. Most of the designs in this course use additional modules to make their implementations have visual indications on the development board.

4.8 Exercise Portal

All materials for the course is presented on Plussa [23]. This portal has multiple features that benefit both course staff and students. The portal is based on Aalto University A+ learning system project. [3]

Firstly, all of the material is available in one place, so that they do not have to be searched from the external sites. Also, if external material should be necessary, there would be links on the portal directly to the required material. Secondly, it provides student with a place where the progress in the course is easily monitored. [2, 3]

Lastly and most importantly, it provides the course Docker based automatic testing environment. This means in practical terms CI like functionality for the logical designs, as the design can be run on the server with ModelSim in a command line mode. Simulation results can be automatically assessed from the VHDL assertion prints. Based on these prints the test can be marked as a success or failure. After the testing has a result, it will be posted to the task grading page on Plussa and the student can gain the grading from the task nearly instantly. This means that the manual grading based on the simulation pictures, which was used on previous implementations, is not required. The role for course staff will be to confirm demonstration tasks and following of the design rules. [2,

3, 18, 31]

While the process of setting up the Plussa system is time consuming, during the course execution it should enable course staff to use their time more efficiently. Additionally, it is easily reusable to later implementations. Furthermore, even if the exercise sets are later changed again, the testing itself will still be necessary and can be done with the same setup.

One of the created functions was exercise type that fetched link to file in TUT's Gitlab [38] for courses. This was used to both easy access to the file, as well as a return commit tracking. Furthermore, this exercise return structure was reused in other course, System Design, to the same purposes. While originally these were not designed as used on courses and files of a multiple kind, they ended up being usable that purpose.

The Docker based system also requires a Docker container to be built to house the actual virtual operating system which is launched when the container is launched. Sharing and reusing were one of the considered aspects when the container was created. For purposes of the digital design course, it was necessary to have Git and ModelSim on top of the basic Linux functions. TUT's VHDL course, logic synthesis, created Plussa portal at the same time as digital design. The container was created together with that course. As a result, both of these courses use the same container to run ModelSim tests. The container is shared from Docker Hub. This simplifies the maintenance process of ModelSim based testing. [12, 13, 23]

5 COMPARISON BETWEEN EXERCISE SETS

This section of the thesis compares the newly created project with the old implementation of the exercises. Each of the set is analyzed in their own section. The main focus is to highlight the changes that are expected to be improvements. Additionally, the rough estimations of the workload for both staff and students are presented.

5.1 Practical arrangements

Lectures on both course implementation were held in both Finnish and English. The same was also for paper exercises. Computer exercises were nominally designated to each language, but in the sessions students could get help with both of them.

Both implementations have their computer exercises held at FPGA class room, that has 21 computers with EDA tool software. The classroom is very vacant during the first period of teaching and during the second period more used. Room used to have 9 old type Altera DE2 boards and for the new implementation it was fitted with 21 new PYNQ-Z1 boards. Additionally, the classroom has teacher's computer which also could be used by students if there was a crowded session. This computer did not have old type board but it too received new board. [24, 28]

There was a possibility of loaning development board with both old and new set. Regardless of this, both sets required the presentation of demonstrations to be done in the class.

5.2 Structure of the exercises

The main structural difference between the projects is the focus on the subject. The new project is all about developing parts into a single system.

The Git version control system was tested for the earlier project but it did not become part of the exercise process. In the new system it is considered a core feature of the project, as it simplifies the testing process in addition to being the actual version control system. In case the student would lose data or it would be altered to work incorrectly, the repository could be used to return the files. Additional benefit with Git comes from the fact that the course is meant to be pair work. File versions between the pairs should be

Table 5.1. *Comparison of the exercise topics and project relevant tasks. All tasks of the new project directly contribute to the greater project.*

Exercise set	Focus of the Old set	Old project tasks (% of the tasks)	Focus of the New set
1	Introduction and simple arithmetic	- (0%)	Introduction and Interfaces
2	Algorithm implementation	Seven-segment controller (25%)	Arithmetic and Hierarchical designs
3	State machines	FSM indirectly (25%)	State Machines
4	Arithmetic Modules and Testbench	ALU and Register bank (50%)	Extending design and Data sheets
5	Pipelining and integration	Integration (25% with possibility of another 25%)	Register redundancy and Integration

automatically synchronized, as each of the tasks require the file to be pushed to the Git repository for return. [16]

The work flow of the exercise is practically identical between the sets. It starts by designing, then simulating and lastly verifying the designs. The Greatest difference in this process is the FPGA synthesis tool, that was replaced.

The main focus of each of the sets in both old and new exercises are shown in Table 5.1. It also details how much of the each of the old exercise set contributes to the main calculator project. More task based differences are explored in their own subsections.

5.2.1 Differences of First Set

In both sets, the first task set is practically identical. They focus on trying out the HDL Designer with half adder. In the new set FPGA implementation is augmented with trying out the ILA system, which has equivalent in the old system, SignalTap II, but it is never used on the exercises of the course. [25]

In the old set after half adder, some arithmetic components are created. In the new project, more introduction is done with trying out interfacing with the obfuscated frame buffer. Additionally, it helps the students to get more familiar with FPGA implementations, as using Vivado is harder than using an integrated task to do the same operations. Use of external application requires the student to understand more about what the design consists of when synthesizing the design. Also, as the frame buffer obfuscation at this point obfuscated the names of the modules, it had additional challenges.

The old set did not utilize these components in the actual project. In the new project, one can use the half adder to create hierarchical designs. Also, with the perspective of

the whole project, it is very important to try out the interface, before moving on to more complex designs that use the very same interface. In both the old and the new project, all of the tasks of this set was deemed mandatory.

5.2.2 Differences of Second set

The new project focuses on implementing a single module on the game logic side of the project using hierarchical design. The old set focused on implementing a binary coded decimal conversion algorithm twice and it had a supplemental sequencer task. Both sets approached the set with a bottom up design mindset. On top of these tasks, both of the projects had debugging exercise as part of this set.

The debugging exercise of the new project was mainly placed here as it was done so in the earlier project. Its placing at this point felt the most useful in the timing of the whole project. Teaching debugging is one of the most important things on course like this and it felt correct to place it in this set. To be precise, it has to be early on course, but not too early so that it is not diminished by lacking skills how to use tools.

In old exercise set half of the exercise tasks were marked as mandatory to do, while in the new set required as minimum 3 out of 7 tasks. These would result in an alien module.

5.2.3 Differences of Third set

The old exercise set focuses on implementing state machines with the state diagram tool of the HDL Designer. This was done by implementing 3 different simple state diagrams and testing simulation delay in one task. On new set, students create additional game modules. It was designed that first module would be implemented with a block diagram and then students would move on to using state diagrams. The block diagram was figured to be the best choice for the first module as it can be implemented with a very similar system than in previous exercise set.

The old set focuses more on implementing various modules with the new diagram type to emphasis its usefulness. The new set has less emphasis on state diagrams. The write sequencer task of this set would later be extended. The old set does not have this kind of task that would later be modified.

Comparing the tasks relevant to the greater project, the only task with the old set that had anything to do with the calculator project, was not directly used in implementing the final product. This is highly contrasted when comparing with new set, which had all of its modules implemented as parts of the game.

5.2.4 Differences of Fourth set

With this set, the old exercises provided variance on how to implement modules. At this point, the implementation diagram was no longer restricted to a specific one. How students would create their designs was now up to their own preference. The new set focused on implementing modules to finalize the game design. These were felt to be easiest to implement with state diagrams, however, it was not restricted how they were done.

In the new project, at this point was division on the tasks that were about game logic and display module logic. Students could pass the set while creating only game modules.

The display controller tasks of this set relied on the state diagram tool. They tried to combine data sheet interpretation to part of the task. There was no such task in any of the old set. Previously, reading data sheets were only part of paper exercises.

5.2.5 Differences of Fifth set

In the old project, this set included an integration task and an additional selectable task which did not have anything to do with the project. In addition to these, there was chance to create one of the earlier parts to the calculator, if the student decided to skip on them earlier. In the new set, tasks focused on a register bank system and a debouncer. The debouncer task was reused from earlier implementation. It was one of the selectable exercises.

In both of the sets, the last task of the last set is assembling the project. In old set there was a base onto which was possible to add the modules created previously in the course. In the new set, the functions of the modules were felt to be more self explanatory, as their purposes were also explained when the module itself was created. As such, the assembling instructions were left very vague. Both assembly tasks try to teach a design hierarchy as the last thing that student would have in their mind in course. This is further emphasized by the new set as it also has the register stack task that focuses on this concept. It is notable, however, that the this is an optional task that all of the students likely will not try to create.

As the old set is possible to be completed without completing the actual greater project, it can feel that the tasks are just a collection of various smaller tasks. All of the tasks in the new project create parts to the whole game project.

5.3 Staff Workload difference estimation

The greatest difference in the staff workload is the introduction of the Plussa portal to automatically assess the returns. While grading any single task is fast and straightforward, the volume of the tasks to be checked is what takes a lot of time from staff. As the course is one of the earlier courses on the computer engineering and embedded systems, it has relatively large attendance rates.

In rough estimation, approximately 70 students each year finishes the exercises with a passing result. Earlier, for the return of each task there were simulation pictures to contain the simulation pass print and the waveform. On top of these, the assessment of the task includes inspecting D&V for schematic. While D&V enables the TA to not open HDL Designer each time, but only simple web page, it still takes some time to inspect all of the diagrams for the design rules.

Some scripts can be utilized to help the checking process, but for thorough checking it still will take up to few minutes in the worst case. This time depends on the complexity of the design. With the volume of students on course, it can be approximated that each of the task checking can take up to two hours. This time includes fetching the returns, grading them and inserting results into the return portal. Demonstration tasks are simpler than this, as they do not require simulation picture inspection, rather they are only checked on the exercise session. Even with those, schematics have to still be checked for design rules.

This estimation assumes the task would be graded one at a time, but usually the grading flow would be to analyze all of the tasks created by single group. This speeds up the grading process as all of the diagrams from single group are available from the D&V at a glance. With Plussa, the grading of functionality automated, and the D&V is practically only checked for design rules, which speeds up the grading process compared earlier implementations. It is still expected to take from hour to 2 hours to inspect each of the visualization results, but as the functionality should be graded from the initial submission, it should be substantial time saving for the staff.

5.4 Student Workload Estimation

The course has had a notorious reputation of being one of the more effort requiring courses. The new set has been figured out to still be on the higher end of time consuming if the student wants to complete all of the various tasks on course. However, it was also intended to have ways to complete exercises without that much effort.

The new set of tasks have more detailed instructions so that students should not feel too lost in doing the tasks. As described in the previous section, staff should have more time to dedicate in other than grading, these include helping with various problems. Even if the

Table 5.2. *Workload estimation of the first and the second exercise set.*

Exercise Task	Load estimation	Learning Target
Half adder design	Light	Introduction to design tool and simulation
Half adder implementation	Light	Introduction FPGA prototyping flow
Interfacing	Light	Introduction to display controller interface
Interfacing with I/O	Light	Introduction to IO with FPGA board
Incrementer	Light	Arithmetic components and hierarchy
Decrementer	Light	Arithmetic components and hierarchy
Left shifter	Very light	Components and wiring
Right shifter	Very light	Components and wiring
Binary converter	Light	Truth table and simple algorithm implementation
Debugging	Medium	Debugging
Alien Module	Heavy	Hierarchy and registers

students find tasks harder to do than what they can cope with, there should be plenty of opportunities and time to ask for help. Pushing the difficulty factor of the tasks can make help teach students to became better designers.

Estimated load and main learning topic of individual exercises are detailed in Table 5.2 and Table 5.3. Exercise loads were tried to be balanced as relatively equal time consuming. It is notable that as the course progresses, the students are expected to become more skilled in designing and hence the complexity of design tasks, and with that the estimated workload, increases.

Minimum requirements for the completed exercises were slightly over half of the available tasks. Approximately for an average student this should be around or slightly less than half of the time it would take to do all of the tasks. Regardless of the workload, if the project and the topic itself are interesting, the students are more willing to spent substantially more time on it.

Table 5.3. *Workload estimation of the third, the fourth and the fifth exercise set.*

Exercise Task	Load estimation	Learning Target
Gun Module	Medium	Hierarchy and registers
Write sequencer module	Medium	State diagram design
Hit detector	Medium	Implementing equality check
Bullet flight modeling	Medium	Adapting designs to new features
Victory effect	Medium	State recognition and visualization
Alien improvements	Medium	Extending existing design
Serial transmitter	Light	Data sheets and state diagrams
Shield reset module	Light	Data sheets and state diagrams
Shield gamma module	Medium	State diagrams
Register bank	Heavy	Hierarchical design
Register bank to serial module	Heavy	State diagrams and register access
Button Debouncer	Light	Modifying IO signals
Assembly	Light	Hierarchical design

6 EVALUATION OF THE FIRST IMPLEMENTATION

This section covers how the projects first implementation fulfilled its intended role on course in the fall of 2018. Results are analyzed from multiple perspectives: Does the tasks themselves function for the intended purpose, do testbenches work for both the designs and automatic grading system and lastly how time intensive the tasks are. Potential issues with the project are explored in detail.

Tampere University engineering courses have Kaiku feedback system which provides feedback how well the students felt the course achieved its purposes. Students were also asked to report in time spent on each of the exercise sets. Additionally, as last task of the project the students were asked to write short feedback on the computer exercises specifically. These were the data used to analyze the exercises.

The content of the new course implementation is depicted in Table 6.1. Course adjusted some of its content but main topics presented remained the same than previously. New exercises were initially figured out to be arranged differently, but this was reverted to be like previous schedule. Main difference in computer exercises topics is removal of delay analysis and incorporating design extensions.

The first section of this chapter explores issues found in the project and each of the exercise sets are explored in their own subsection. After this there is evaluation of time that both staff and students have had to invest in the course. Lastly the most significant results are concluded. These results will be used to plan the potential improvements and tweaks in the next chapter.

6.1 Functionality of the Implementation

The project itself was found out to be interesting, but it certainly was not perfect nor implemented perfectly. The general structure of the course was felt to be appropriate to teach the goals of the course to the students. Lectures provided the concepts of topics, paper exercises introduced the concepts and lastly computer exercises introduced to in practice implement the systems. Tasks themselves were interesting and challenging.

Among other things, exercise tutorials need refurbishment. They have great amount of useful instructions, but they are also scattered to general design rules and to computer exercise tutorial pages. This makes the amount of information initially presents too much.

Table 6.1. *Course contents with the new exercise project.*

Lecture Topic	Description	Ex #	Exercise topic	Description
Motivation and course arrangements	Teaching tools, materials		Introduction to tools and systems	questionnaire about practices
Digital design flow and views	Design process and automation Design abstractions and models		HDL Designer and ModelSim Tutorials	questionnaire about practices
Specifying combinational logic	High and binary level specification Gate networks			
Designing combinational logic	Minimizing switching expressions Standard combinational modules	P1	Designing combinational logic Data sheets	CMOS transistors, car cabin light diagram, RAM data sheet analysis
Analyzing and optimizing combinational logic	Analysis of ready made gate network Design multi-output gate networks	P2	Analyzing combinational logic	Critical path, ripple adder, delays block diagram
Specifying sequential systems	Time functions, sequences, Mealy and Moore state machines, state transitions	C1	Hello LED on PYNQ	Learn to use design, synthesis, simulation and FPGA programming tools
Designing state machines	Canonical state machine Standard sequential modules	C2	Alien module	Arithmetic components, hierarchy, registers
Time behavior of FSMs	Timing, state register, signal paths, delay analysis, determining clock frequency, clock skew	P3	State tables and diagrams	State machine diagram, State machine block diagram, sequence recognizer
Analysis and features of state machines	Analysis, registered, one-hot, HDL designer	T3	Gun and bullet modules	State machine design
Arithmetic Components	Addition of integers, ripple-carry, carry lookahead, and parallel prefix adders, signed numbers and their addition	P4	Analyzing sequential syste	Delay, timing, and critical path in sequential systems, setup time, hold time, and propagation delay in D-FFs
Arithmetic: Multiplication, standard modules	Multiplication, shift registers, accumulators, ALU	C4	Make the game objects	Extending designs, Data sheets
Programmable logic circuits, RTL design	FPGA architecture, history and future, control/data architecture, shared resources, abstraction	P5	Misc. topics	Pipelining, resource sharing, ALUs, RTL design
RTL analysis	Analysis of data/control paths, data memory, FIFO, buses	C5	Complete game	Hierarchical register structure Combining all of the blocks, debounce register redundancy

This results in difficulties trying to process all of it. As some of the information is only used later in the course, it also creates some confusion in the student. Additionally, some of the students do not tend to return to restudy the material presented. All in all, presenting the material as it was on this implementation, was not optimal.

Few of the course testbenches did not function precisely as they were intended. These issues arise from the reference project implementation style choices. Testbenches follow too closely what the base project does and with slight adjustment in task instructions this causes verification not to work to system students instructed to create. This highlights that there was not enough testing done with varied enough specification fulfilling solutions.

6.1.1 Issues with First Set

First set had a few minor issues which caused some confusion in the students. These were not great flaws, but they are something that should be fixed to have the start of the exercises feel better. From the grading point of view this set had two demonstration tasks, which made the set awkward to take demonstrations from.

In the half adder task FPGA implementation part, few of the steps required clarification. These included changing the bitstream file on programming the FPGA after the first part and inserting the design into another top level diagram.

Last two tasks of the set had some clarification need to their description. Additionally, they are very similar tasks, and both have in them the part that student should synthesize the design for FPGA board. Synthesizing instructions have the student connect the design to top level which has the obfuscated display controller module, but it lacks the precise instruction list what files add to Vivado to be able to synthesize the module.

6.1.2 Issues with Second Set

The second set flaws were mostly in instructions. Refining those should be the most important improvement on this set. One of the testbenches, for the alien task, did not work as intended.

The shifter tasks in second exercise set did not have precise enough restrictions. This meant that, by searching enough, it was possible to find an arithmetic module that did the shifting. The purpose of the task was to implement the shifting with the combinatory and bit manipulation mechanics.

The debugging exercise lacked the specific instructions to program the design to the FPGA board, which lead to some confusion about verifying the functionality. Additionally, the lack of a testbench, while intentional, did not benefit the goals of the task.

The alien task testbench had defective assertion mechanic which made it harder to use

than necessary. In practical terms, the testbench required clock accurate specific positions for the target coordinate, while it was enough to have them slightly off and still produce good enough for purposes of the game project. Additionally, this task was a demonstration return, so simulation termination did not have any advantages. Instructions were good enough to understand the target system, but it was generally necessary to have some hints how to start implementing the design.

6.1.3 Issues with Third Set

Third set had only minor issue with hit module instructions, but greater issues with testbenches. These were on the write sequencer and the hit module tasks.

The write sequencer task was designed to have only a generic testbench for testing the design. More precise functioning testbench would be required, as students are not yet fully familiar with using the display controller that the module is supposed to communicate with. Additionally even if they figure out the various signals perfectly, it is an unproductive process to continuously drive the signals manually as the sequencer needs to later be improved.

The hit detector task had a defective testbench. While very simple coordinate comparison is all that is required to create the desired functionality, the reference design game uses a more complex system. This made sure that it was not possible to gain more than one hit detection with one successful hit. It was done by checking for the bullet reset value. As the task did not specify how the module should precisely function, the testbench worked incorrectly. Test provided correct waveform from the simulation, but it did not recognize the correct sequence of hits. The test can be simplified while keeping the intended functionality. This also meant that even though automated testing was meant to be used in this task, it needed to be graded manually. In addition to manual grading, the task return system needed to be altered after it was already open to students.

6.1.4 Issues with Fourth Set

Fourth set had most of its problems with the gamma setting module. The task was deemed to be too hard to design properly with technical knowledge accumulated this far. Additionally, the used HDL Designer state diagram functionality requires students to use VHDL operators. Without knowledge of these it is very difficult to for example create loops and indexing which were in practical terms necessary to use with this module.

Two additional lesser issues were also with the exercise set. These were with the serial transmit module and the reset module tasks. These issues were problematic as the task was to study data sheets.

The serial transmit module testbench did not allow enough fast serial clock signal for

moving the data to the shield. Testbench was designed in a way it could not receive signals as fast as DM163 microcontroller in the constant current led driver shield allows. Students were instructed to avoid exceeding the testbench limitations. [7, 11]

Another data sheet detail was misinterpreted in the reset module task. In reference project the reset sequence was 1-0-1 as it in actuality was simply 0-1. It simply required enough many cycles to be low to reset fully. This was solved by modifying the instructions to instruct students to implement the sequence that was used for the reference project. [7, 11]

Both of the data sheet task problems required the task instructions to be updated while they were already released. In both cases, the correct functionality was explained to students and adjustment to those values instructed.

6.1.5 Issues with Fifth Set

Fifth set had its issues too. These were mainly due to lack of design instructions combined with complex design.

Register bank design was too large as one task. Individual modules within it are usable design tasks that teach important concepts. Also, the register bank to serial data module was highly time consuming to create. This is very similar issue with the gamma module presented in the previous section. The student might also at this point be familiar with VHDL from the logic synthesis course [21], which creates large advantage for understanding the various operators one could use with the diagrams.

Additionally, the combined system was not tested enough, as nearly all of the attempts to create the display controller failed to create a working system. This was generally because of the bank to serial module or the serial transmitter module. By replacing either one of these modules with a premade one, students would get the rest their system working. This means that there is some allowed variance in the task descriptions that create issues when the modules are working together. Also, assembly instructions were completely omitted, and as such, it was not distinct which of the modules were meant to be connected to each other. This was oversight on designing the task as it was figured out that as each of the modules were explained in enough detail in their own tasks.

Another issue arose as the task required students to replace obfuscated design with their own created module or one with non obfuscated block names. This was due to mismatch with top level port names with the custom constraint file provided by the course. This caused errors with the synthesis process on Vivado. While these were somewhat easily pinpointed by staff to the constraints, it was something that was not taught or instructed anywhere. This lack of instructions was by design to avoid students using pin configuration features, as there is high chance to cause errors with that approach.

6.2 Staff Load Analysis

As expected, omitting manual grading from staff eased the workload created by the course. It allowed them to focus the time on helping, grading design rules and other tasks.

It is difficult to measure the difference in time saved by the automatic grading as exercises were no longer the same. As one of the automatic grading tasks did not function correctly, it had to be graded manually. This gives some insight to how much time was saved. No helpful assisting scripts were in place for this process. This grading took approximately two hours for all participants. With help of relevant additional checking scripts it is likely that this could have been halved. If the course would have fully manual grading, it would be likely that some local helpful automation would be created and that makes measuring the difference directly not accurate.

As there were 18 automatically graded exercises, this means that the process likely saved at least around 20 hours of grading work during the course. As these hours are used up during the course implementation, it would mean that the staff would not be available to other tasks during it.

If done to great lengths at a time, the manual grading process becomes error prone. Automation removes this human aspect of the equation, too. Additionally, as the student gains the grade automatically, they do not have to wait the task to be graded and feeling of instant feedback is satisfying. It also removes the uncertainty whether the task will function on grading in a way that student intends it to.

6.3 Student Load Analysis

As computer exercises are only part of the course, exercise load should not be too high. Effectively, the course load distribution within the course is the highest part, but it should not be high enough to be taking of all the hours dedicated to the course. It should be noted that while students were to report their individual hours, most students participated in the course exercises as pairs. Reported hours were gathered and analyzed after the course ended. Results for each of the sets are presented in Figure 6.1. From this figure, it can be seen that each of the sets has some students reporting very high values.

While individual hours point out problematic cases in each of the set, highlighting key values for each of the tasks and calculating accumulation of these can yield additional insight. Results for this are presented in Figure 6.2. It clearly shows that the worst case results far exceed the planned time for the whole course which is 125 hours. It should be noted that this combines the hours of each of the worst case. However, average and median values are relatively similar which indicates that most of the students reported to spending around that much time. These values are too high for the considered role of

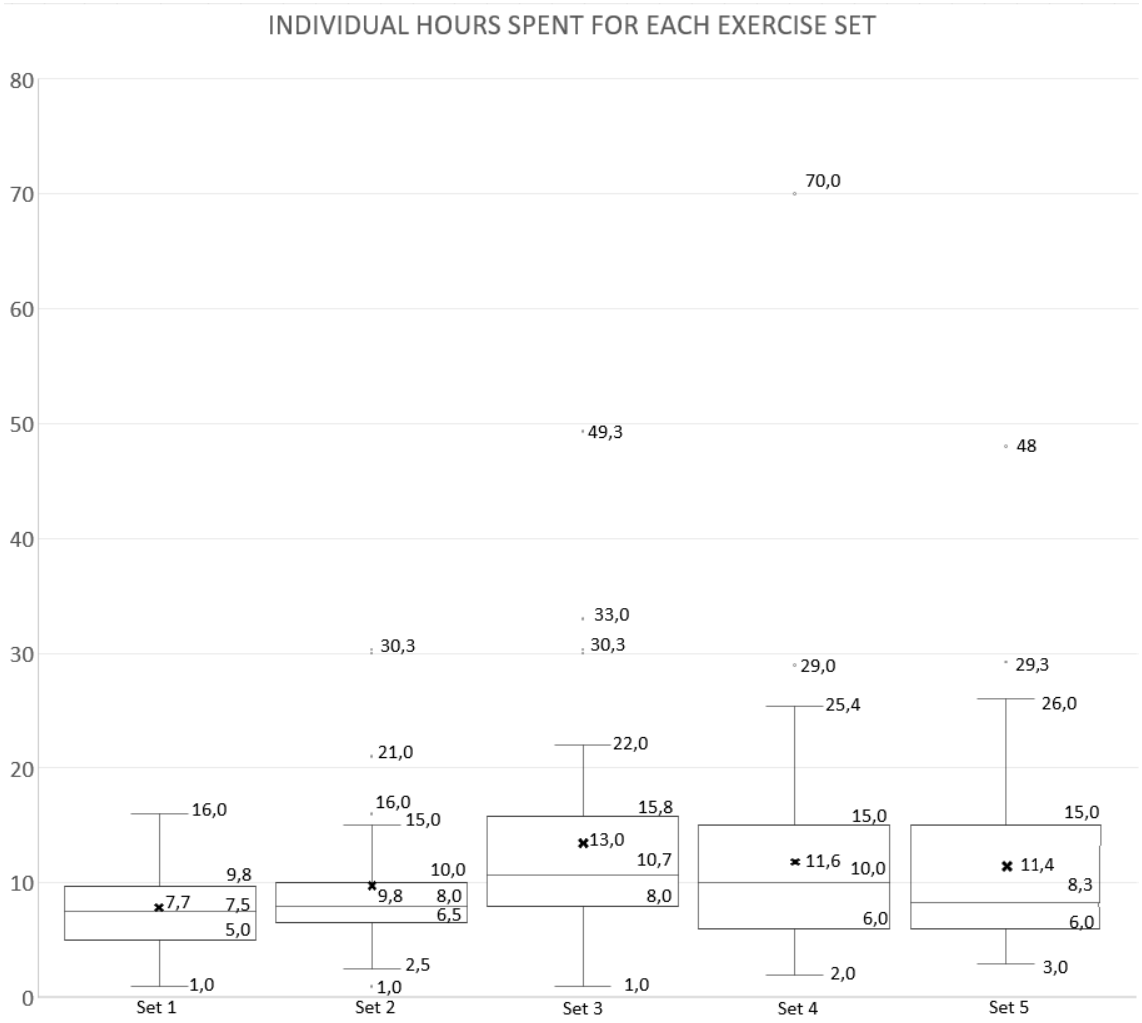

Figure 6.1. *Reported individual student work hours for each exercise set.*

computer exercises.

Another value that was looked into was reported hours which were normalized to be compared as if the students had kept up the pace and done all of the exercises. Values for these are presented in Table 6.2. These results additionally highlight how time consuming the set was for various types of students. Especially, if the worst case is considered, it highlights that those that did have great issues with exercises, did fewer exercises than the ones that were near average values. This indicates that it is very likely that students were forced in a trial and error approach, and that they were not able to consult staff. The Bloom's taxonomy explores this approach, too. In two dimensional interpretation of the taxonomy, this means that student might lack the knowledge of how to apply various concepts that were presented in the practical work. It is worth analyzing further how various parts of the course interact and ready students to the computer exercises, but that is out of the scope of this thesis. [14]

It is also worth noting that students are of varied background and nationality. Way the exercises are phrased and designed might be more obvious to certain types of students. The minimum case of spent time presented in both Figure 6.2 and Table 6.2 is fewer

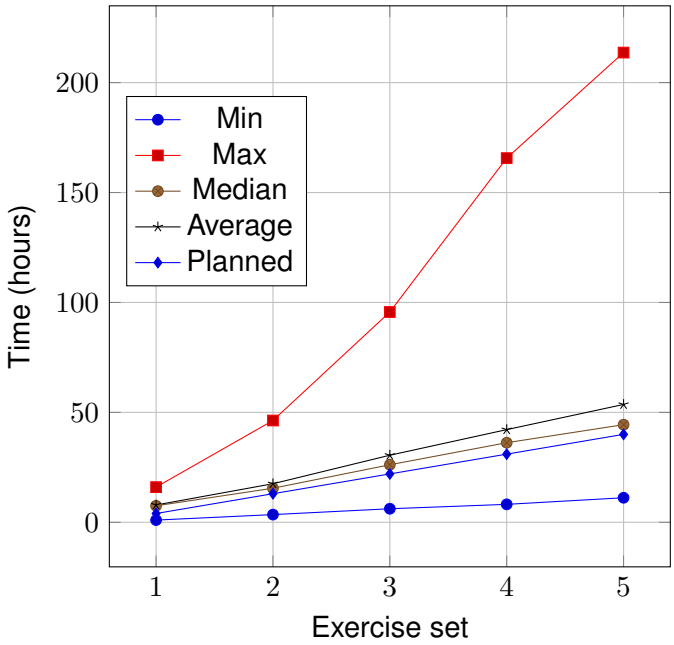

Figure 6.2. *Accumulation of work hours during exercises.*

Table 6.2. *Exercise times normalized to full points*

Sum of	Reported hours	Normalized hours
Median	44.4	56.7
Average	53.6	74.6
Min	11.2	16.5
Max	213.7	384.0

hours than was figured out to be likely with the new set.

6.4 Results Conclusion

Despite the flaws in the instructions and tasks themselves, students found out that the project was interesting. Hands-on feeling from constantly implementing the game on development board was something that was well received. However, it did also drew criticism as it makes students have to wait for synthesis result each time.

Large time investment by students did not alter the portion of students that passed the exercises. The last old exercise version of the course had 65 % of its original participants pass the required amount of tasks. With the new exercises, this same value was 67 %. From this it can be concluded that students that originally planned to take the course, were willing to invest in completing the course, even as they got the feeling of an increased workload. It could be argued that greater amount of design work with computer exercises made the students more skillful designers, however the student grade distribution on

Table 6.3. *Perceived workload according to Kaiku feeback*

Perception of workload		Low	Appropriate	Too high
Year of implementation	students	%	%	%
2017	47	2	60	38
2018	48	0	42	58

course in comparison with previous years did not differ enough that it would be possible to claim that or the opposite.

One of the most important form of feedback gained on the Kaiku system is the perception of the workload by the students. Previous implementation is compared with the new in Table 6.3 and there is a notable increase in the feeling of perceived workload on course.

As the exercise time investment reported by students does match with this, it is quite obvious that the exercises as they are now, consume on average too much time. It is notable, however, that it is just a perception of the load. This varies from person to person greatly. The intended time usage of computer exercise parts of the course has previously been approximately 40 hours. The design process of the tasks was likely too focused in a task to task implementation and it did not properly take in to account the accumulation of the hours spent. As current average used hours reported exceed the target estimation by approximately 30 % some measures are needed to be taken. Improvements and restructuring of the exercises are explored in the next chapter. [8, 27, 37]

7 IMPROVING THE EXERCISE PROJECT

While there are effectively nearly limitless ways how to improve any exercise project and its tasks, in this section thesis suggests a few ways to do it. While a couple of the tasks had testbenches that were not functioning properly, this section focuses on the tasks themselves. These mainly include instructions and their division into the logical parts and design of the modules.

The section first covers suggestions to improve tutorial content and then it points out improvements to each exercise sets. After this section explores ways the possible restructuring could be made. Lastly it suggests a schedule for implementing the improvements.

Terminology used on course should be standardized. The display controller, for example, was called also the frame buffer, the display module and with an obfuscated black box name. This lead to confusion and inconvenience. Additionally, it could be useful to have a general glossary on the web portal, as the course is one that introduces the student to digital logic design specialization.

As an addition, each set's documentation return could be improved to have answers to a few short questions about each of the tasks. These would be used to summarize the weekly exercises and the tasks that were accomplished. Naturally, it would be required to only answer the questions that were about the modules that students implemented.

7.1 Tutorials

The tutorial pages were mostly directly adapted from earlier material with a few modifications. Pages themselves contain a lot of useful information but it is not in easily understandable format. Rewriting guides and tutorials would be helpful to students. Recreating the tutorial images would also be beneficial, as reusing the earlier images can be of older versions which would be incorrect in comparison with the current versions of used software. Additionally, it can help pinpoint more of the useful features from the EDA tools.

Tutorial pages could be distributed along the course to have them introduced where the containing information is also used. This would avoid the students to be overwhelmed by too much information at once and it can help the students to concentrate on the more useful parts of the information. This would make each of the exercise sets conform more to the embedded system taxonomy matrix. [14]

Additional tutorial content could be created to cover the VHDL operator usage with the state diagrams of HDL Designer. In addition to these, loop structure could be made as part of the material. These instructions would make designing with the state diagram tool easier and help students to create more complex state machines.

One of the tutorials which was added, was about simulating design without a testbench. This is a useful concept but it should likely be only additional information on this course and not required for the completion of tasks.

7.2 Exercise Task Improvements

Various tasks that have clear improvement or restructuring are presented in their own set's subsections. Additionally, nearly all of the exercises could use additional clarification to their instructions. All tasks that require using the obfuscated display controller should be recreated to use module versions that have correct diagram names and all of the files needed to be added to Vivado should always be listed at the exercise.

Most of the testbenches should be added timeout function as it would clarify the error messages both locally and on Plussa. As of the previous released version, if testing becomes deadlocked and it runs endlessly, it will eventually get terminated by the Plussa server. This interrupts the grading process and it will leave the student without feedback. This would also be the waste of server resources while it operates as stuck.

7.2.1 Improvements to Tasks of First Set

In addition to improving the half adder on FPGA task description, two interfacing tasks should be restructured. Half adder task could be changed from demonstration to submission of submission of an ILA screenshot to manually confirm that student has looked in to FPGA implementation. Interfacing tasks should be combined to single design task and single FPGA part. This would mean better separation between simulation and FPGA implementation parts.

These changes could help teach students some hierarchical design right away and simplify demonstrations. Additionally, it would reduce the times student create FPGA synthesis, which can reduce student downtime when doing the exercises.

7.2.2 Improvements to Tasks of Second Set

The alien task of second set was simply too large as it is. Splitting the task to hierarchical modules, each worth of some exercise points would likely solve the issue. Separation of these modules would result in having three modules: the coordinate system, direction

control and health control.

The coordinate system would be hierarchical design having register to save the coordinate. It would have a control signal to proceed to the next coordinate and direction selection in addition to the clock, the reset and data signals. Depending on how the course is progressing, control signals could be made with canocial state machines. The health control system would be basically place holder for the upgraded health system, but it would still need register to hold the current status of the alien.

If the split would be done like this, in addition to fixing the existing top level testbench, each of the submodules would need their own verification testbench. Regardless of how the split is done or if the instructions are simply improved, it would be important to evaluate how it takes to make the module.

The debugging exercise of the set could be changed to be part of other set, as this one is already quite extensive. Furthermore, manual simulation of design does not bring any advantages so it would be beneficial to remove that aspect of the task. It would be very useful in a task like this to have the very precise description of the target system that would be working. It is hard to start debugging new design that one has not seen before, if its purpose as a functional module is not understood.

7.2.3 Improvements to Tasks of Third Set

The testbench for the write sequencer module should control various interface signals. Additionally, it would be good to have event notifications when it allows a certain module to transmit their display data to help student focus on the important moments.

One of the possible changes is combining the previous set's debugging task and gun module tasks to single exercise. It could consist of the student copying a given diagram to their own library, and then improving and extending the design.

There should be an additional note in the tasks of this set that with which diagram style they should be implemented with. It might not alter the choices that the students make but presenting the choice helps the student to be clear that which are allowed.

7.2.4 Improvements to Tasks of Fourth Set

Most flawed task in fourth set is the gamma module task. It simply requires too many design specifics of the VHDL language to be doable for a student in reasonable time. Presenting better instructions along with an operator tutorial could make it a reasonable task as it is.

The testbench for the reset module should incorporate the correct sequence and it would also be helpful track the timing of the reset. Also, the serial transmitter testbench should

be fixed to enable the correct range of transmitting clock frequency. This should also be tracked with some kind of counter.

7.2.5 Improvements to Tasks of Fifth Set

The register to serial task has the same flaws as the previous gamma module task. Without knowledge of operators, students could circumvent their usage with creating the functionality without loops. This results in a state machine which has over 64 states compared with a loop structure that has only relatively low amount of states. As with the gamma module, an operator tutorial could make the task more feasible as it is.

The register bank task was good in teaching a hierarchical design, but it was too complex to be one task. It could, similarly than the alien task of the second set, be split in multiple tasks that all have their own task and tests. These sub tasks would be a single register module, a column module, a bank module, a bank selection controller and finally bank top level. Some of these could be given as ready made parts if the amount of modules and therefore tasks is deemed too high.

7.3 Improving Exercises without Restructuring

Perhaps the simplest improvement of the project would be only to improve the existing tasks to accommodate previously mentioned improvements. Additional minor changes could be made to have the tasks to be changed from fully designing the module to extending existing an existing design, adapt or debug existing diagram types to project purposes. These types of tasks could also be closer to a practical project as it is rare to have modules are started from nothing.

This approach relies on the existing tasks and the exercise structure being good enough with task based improvements. If exercises would be too short with improvements, some of the modules could be extended to have additional features. Alternatively, if exercises are deemed too time consuming even with improvements, some of the features could be reduced.

7.4 Partial Usage of Display Controller

Some of the display controller modules could be given as ready made, and some as tasks. New partially excluded display module architecture is shown in Figure 7.1.

Partial usage could enable a prototyping type task with the gamma values of the shield. The task would would need a reset module, a transmit module and a static one transmitting serial controller. After successful gamma simulation the student would be tasked to

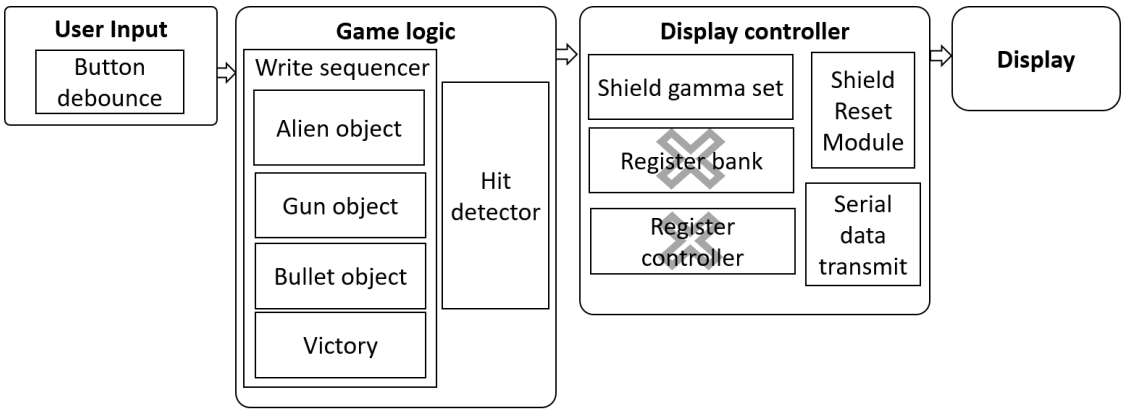

Figure 7.1. *Project architecture with some of it's display controller parts removed.*

alter the gamma values until they get white colour on LED matrix. This was an idea that was tried to be implemented originally, but it ended up being not done as the creation of the FPGA implementation required all of the display modules. The idea was to the tweak gamma values after the game was assembled fully, if the student did not like the colors.

The register to serial module and parts of the register bank are most taxing parts to create, so it would be logical to have those provided as ready modules. This would enable rearranging the rest of the tasks to create more relaxed schedule. The gamma module task is likely to be most time consuming part of the display controller if this approach is taken.

It is likely that creating additional tasks to extend the exercise project would not be necessary as the project is quite time consuming. However, it might also be needed as the structuring of the existing exercises is improved and likely will take less time.

7.5 Removal of Display Logic Tasks

The most radical way to improve the exercise set would be to simply separate the display logic module as a course given module. This architecture is presented in Figure 7.2. This solution would mean that it would be necessary to create additional game logic modules or features to complement the removal.

This approach would also mean that display module could be given without obfuscation. This would have added benefit of students being able to look how various parts to it are made. This would give them potential to draw inspiration from or it could be turned in to a task that explores how the parts are made. This type of implementation could also have the task of exploring gamma values are set.

New tasks could be additional features or extensions to existing modules. Additionally, more of the FPGA implementation tasks could have additional ILA tasks to verify signal contents within implemented design running on the board.

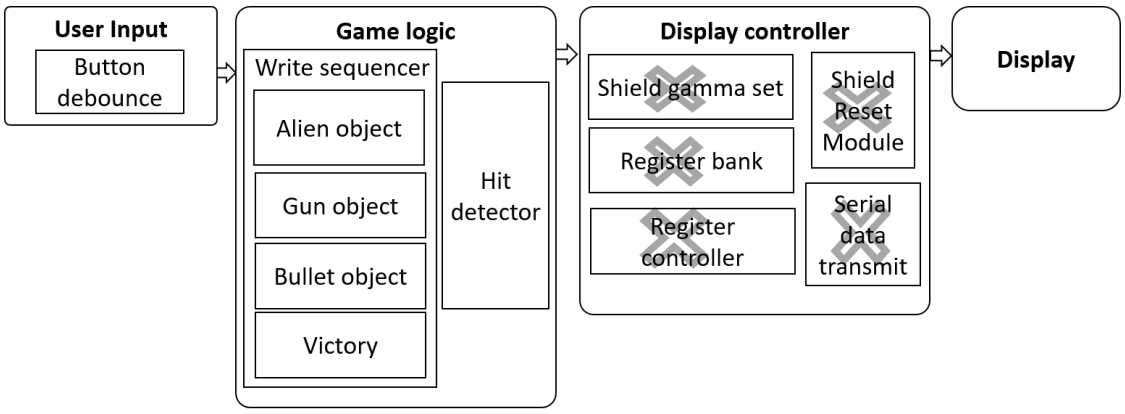

Figure 7.2. *Reduced task architecture with removal of display modules.*

7.6 Graduated Approach

Exercises could also try to incorporate graduated exercise project. This would use all of the versions of the exercise restructuring options. These would be presented to students as a potential completion choices for the students. Simplest approach would be aimed at those that do not intend to continue with digital design, while the most complex and time consuming would be aimed at those students that intend to become experts on the field.

This kind of approach is already in use on some of the programming courses. For example, in introduction to programming the requirement to being able to attend the next one, is built within the course grade. Choosing the exercises according to personal goals can motivate the students to try to progress their skills or concentrate on other subjects while still gaining passing grade. [1]

Exact limits of the exercises and the progression of studies would need to be considered in greater detail if this approach is selected.

7.7 Schedule for the Improvements

The exercise project as it is provides a good and challenging set of tasks to do, however, it is too time consuming to be kept as it is for another implementation. This was highlighted by the student responses and statistics as the most pressing issue. Other issues were relatively minor and could be considered part of consuming the time or, at least indirectly, contributing to it.

This gives the improvements an easy scheduling of being finished when the next course implementation starts. While the course does not start with computer exercises, the need to distribute version control repositories for students makes it the deadline for fixes and restructuring of the files and hence the tasks.

While this could be circumvented via providing the students with the files later, it would

make the already relatively time consuming and tricky file management even more complex. Hence it is not considered a viable option. However, the actual task descriptions and tutorials can be improved up to their release date, which will be later on course. While it would not be considered good practice or optimal, that is the extent of time which the project could still be improved. Most pressing matter is selecting the direction exercise improvement will take.

8 CONCLUSION

This thesis documented the development of the new computer exercises for digital design course. The project included the implementation of the reference exercise project and creation of new exercise tasks. The thesis highlights aspects of implementing new exercises and adapting new work flow. New exercises attempts to make the course more useful to students.

The thesis started by exploring previous structure of the course and highlighted issues with the old exercises. The main points of these were aging development boards, their specific required EDA tools and fragmentation of the exercise set.

The new FPGA board was taken as the platform on course and at the same time it was decided to modernize the exercise project. The exercise porject was decided to be modeled after Python programming course task. This was decided as to highlight various methods of producing the same end system as well as it was figured out to be interesting for the students. Along with exercises, the new exercise portal system, Plussa was taken to use. This platform enabled the course to gather all its material and exercises in one portal and use automatic grading process to grade the functionality of the tasks.

The project was found out by the students to be interesting, but after the first implementation it was discovered to be too much time consuming. The grading system was implemented successfully, and it saved staff time that was previously used for grading the exercises. Some of the testbenches created for the course were the following reference project too closely and it caused issues for the tasks they were associated with. The time investment need for students was a result of combination of unspecific design and lack of specific instruction steps that were deemed important in addition to tasks being more complex than previously.

Lastly the thesis explored ways to implement improvements to the exercises and suggested a schedule for implementing them. As a whole, the new exercise project was successful and was good base for future course implementations.

REFERENCES

[1] *3.9.2 PYNQ Ampumispeli | Johdatus ohjelmointiin | Plussa. Plussa Course Archieve*. URL: https://plus.cs.tut.fi/johoh/K2018-fi/kierros03/pynq_napit_kytkimet/pynq_lasergun/ (visited on 04/05/2019).

[2] *A+ Course Template*. URL: https://github.com/apluslms/course-templates (visited on 03/31/2019).

[3] *A+ LMS. The extendable learning management system*. URL: https://apluslms.github.io/ (visited on 03/31/2019).

[4] *Arduino Uno Rev3*. URL: https://store.arduino.cc/arduino-uno-rev3 (visited on 04/05/2019).

[5] P. J. Ashenden. The Designer's Guide to VHDL (Third Edition). In: ed. by P. J. Ashenden. Third Edition. Vol. 3. Systems on Silicon. San Francisco: Morgan Kaufmann, 2008.

[6] *Chicken game*. Nov. 20, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h10_t5_chicken.html (visited on 03/23/2019).

[7] *ColorsShield. Magic RGB LED matrix driver shield*. Sept. 3, 2010. URL: https://www.itead.cc/wiki/Colors_Shield (visited on 03/31/2019).

[8] *Course Implementation - POP. Course implementation TIE-50106 2018-01*. URL: https://poprock.tut.fi/group/pop/opas/toteutuskerrat/-/toteutuskerta/2018-2019/75104 (visited on 04/19/2019).

[9] *Cyclone II | Cyclone II Support*. URL: https://www.intel.com/content/www/us/en/programmable/products/fpga/cyclone-series/cyclone-ii/support.html (visited on 04/05/2019).

[10] *Debugging exercise*. Oct. 2, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h04_t2_hissi.html (visited on 03/24/2019).

[11] *DM163. 8x3-CHANNEL CONSTANT CURRENT LED DRIVER*. Aug. 19, 2005. URL: http://www.siti.com.tw/product/spec/LED/DM163.pdf (visited on 03/31/2019).

[12] *Docker*. URL: https://www.docker.com/ (visited on 03/31/2019).

[13] *Docker Hub*. URL: https://hub.docker.com/ (visited on 03/31/2019).

[14] U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. M. Thompson, C. Riedesel, and E. Thompson. Developing a Computer Science-specific Learning Taxonomy. In: *SIGCSE Bull.* 39.4 (Dec. 2007), 152–170. ISSN: 0097-8418. DOI: 10.1145/1345375.1345438. URL: http://doi.acm.org/10.1145/1345375.1345438.

[15] D. D. Gajski. *Embedded system design: modeling, synthesis and verification*. Springer, 2009.

[16] *Git*. URL: https://git-scm.com/ (visited on 04/19/2019).

[17] *HDL Designer - RTL Design Analysis, Assesment, and Visualization - Mentor Graphics*. URL: https://www.mentor.com/products/fpga/hdl_design/hdl_designer_series/ (visited on 03/31/2019).

[18] *ModelSim ASIC and FPGA Design - Mentor Graphics*. URL: https://www.mentor.com/products/fv/modelsim/ (visited on 03/31/2019).

[19] *Moodle2*. URL: https://moodle2.tut.fi/?lang=en (visited on 04/14/2019).

[20] *Opinto-opas 2017-2018. TIE-50100 Digitaalisuunnittelu, 5 op*. (Visited on 04/14/2019).

[21] *Opinto-opas 2017-2018. TIE-50206 Logic Synthesis, 5 op*. (Visited on 04/14/2019).

[22] *PADS PCB Design Software - Mentor Graphics*. URL: https://www.pads.com/ (visited on 04/14/2019).

[23] *Plussa*. URL: https://plus.cs.tut.fi/ (visited on 03/31/2019).

[24] *Program your FPGA in Python - PYNQ Development Board - Digilent PYNQ-Z1*. URL: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/ (visited on 03/31/2019).

[25] *Quartus II Handbook. Volume 3: Verification*. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii53009.pdf?wapkw=signaltap+ii (visited on 04/17/2019).

[26] *Quartus II Web Edition (service packs)*. URL: https://www.intel.com/content/www/us/en/programmable/downloads/software/service-packs/sps-web.html (visited on 04/05/2019).

[27] E. Salminen and T. D. Hämäläinen. Teaching System-on-Chip design with FPGAs. In: *Proc. of FPGA World* (2013). URL: http://www.tkt.cs.tut.fi/Opetus/Fpga_board/Salminen_TeachingSystemOnChipDesignWithFpgas_FpgaWorld_2013.pdf (visited on 04/05/2019).

[28] *Terasic - Phased Out - Main Boards - Altera DE2 Board*. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=30 (visited on 04/19/2019).

[29] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design. Calculator project*. Aug. 29, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/taskulaskin/taskulaskin.html (visited on 03/24/2019).

[30] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design. Important design rules*. Sept. 21, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/suunnittelu.html (visited on 03/24/2019).

[31] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design. Tutorials*. Aug. 29, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/taskulaskin/taskulaskin.html (visited on 03/24/2019).

[32] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design, Computer exercise 1 / Fall 2017*. Sept. 25, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h03_t1.html (visited on 03/24/2019).

[33] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design, Computer exercise 2 / Fall 2017*. Oct. 2, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h04_t2.html (visited on 03/24/2019).

[34] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design, Computer exercise 3 / Fall 2017*. Oct. 23, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h06_t3.html (visited on 03/23/2019).

[35] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design, Computer exercise 4 / Fall 2017*. Nov. 6, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h08_t4.html (visited on 03/23/2019).

[36] *TIE-50100 Digitaalisuunnittelu / TIE-50106 Digital Design, Computer exercise 5 / Fall 2017*. Nov. 20, 2017. URL: http://www.tkt.cs.tut.fi/kurssit/50100/Harjoitukset/h10_t5.html (visited on 03/23/2019).

[37] *Toteutuskerrat - POP. Toteutuskerta TIE-50100 2018-01*. URL: https://poprock.tut.fi/group/pop/opas/toteutuskerrat/-/toteutuskerta/2018-2019/75103 (visited on 04/19/2019).

[38] *TUT Course Gitlab*. URL: https://course-gitlab.tut.fi/ (visited on 03/31/2019).

[39] O. Vainio, E. Salminen, and J. Takala. Teaching Digital Systems Using a Unified FPGA Platform. In: *in Proc. of BEC* (2010). URL: http://www.tkt.cs.tut.fi/Opetus/Fpga_board/Vainio_TeachingDigitalDesignUsingFpga_Bec_2010.pdf (visited on 04/05/2019).

[40] *Vivado Design Suite*. URL: https://www.xilinx.com/products/design-tools/vivado.html (visited on 03/31/2019).

[41] *Wayback Machine, Internet Archive. Binary to BCD Converter*. Jan. 4, 2004. URL: http://web.archive.org/web/20080529051456/http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary_to_BCD/bin_to_BCD.html (visited on 03/24/2019).

[42] *Zynq-7000 SoC. Technical Reference Manual*. July 1, 2018. URL: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf (visited on 03/31/2019).